HACK X CRACK: TCP / IP -- HERRAMIENTAS BASICAS

PC PASO a PASO

VIRUS
ACERCATE A ELLOS

3 SERVIDORES ON LINE PARA TUS PRACTICAS DE HACK

NUMERO 18

HACK X CRACK - HACK X CRACK - HACK X CRACK







PHP
PASO DE VARIABLES





TCP / IP: UDP
PROTOCOLO DE DATAGRAMAS
DE USUARIO



LOS MEJORES ARTÍCULOS GRATIS EN NUESTRA WEB

PC PASO A PASO: CREANDO FORMULARIOS CON PHP !!!





EDITORIAL: EDITOTRANS S.L.
C.I.F: B43675701
PERE MARTELL Nº 20, 2º - 1ª
43001 TARRAGONA (ESPAÑA)

Director Editorial
I. SENTIS
E-mail contacto
director@editotrans.com
Título de la publicación
Los Cuadernos de HACK X CRACK.
Nombre Comercial de la publicación
PC PASO A PASO
Web: www.hackxcrack.com
Dirección: PERE MARTELL Nº 20, 2º - 1ª.
43001 TARRAGONA (ESPAÑA)



Director de la Publicación
J. Sentís

E-mail contacto
director@hackxcrack.com

Diseño gráfico:
J. M. Velasco

E-mail contacto:
grafico@hackxcrack.com

Redactores
AZIMUT, ROTEADO, FASTIC, MORDEA, FAUSTO, ENTROPIC, MEIDOR, HASHIMUIRA, BACKBONE, ZORTEMIUS, AK22, DORKAN, KMORK, MAILA, TITINA, SIMPSIM... ... ...

Contacto redactores
redactores@hackxcrack.com

Colaboradores
Mas de 130 personas: de España, de Brasil, de Argentina, de Francia, de Alemania, de Japón y algún Estadounidense.

E-mail contacto
colaboradores@hackxcrack.com

Imprime
I.G. PRINTONE S.A. Tel 91 808 50 15

DISTRIBUCIÓN:
SGEL, Avda. Valdeparra 29 (Pol. Ind.)
28018 ALCOBENDAS (MADRID)
Tel 91 657 69 00 FAX 91 657 69 28
WEB: www.sgel.es

TELÉFONO DE ATENCIÓN AL CLIENTE: 977 22 45 80
Petición de Números atrasados y Suscripciones (Srta. Genoveva)

HORARIO DE ATENCIÓN: DE 9:30 A 13:30 (LUNES A VIERNES)

NUMERO 18 -- PRINTED IN SPAIN
PERIOCIDAD MENSUAL
Deposito legal: B.26805-2002
Código EAN: 8414090202756

EDITORIAL

EL TIEMPO Y LAS PROMESAS

Grandes cambios se han sucedido en las últimas semanas en este país. No, no me refiero al cambio de gobierno, eso es lo de menos… lo importante son los cambios sociales, esos que hacen de nosotros quienes realmente somos.

Hace tiempo, en el Foro de Hack x Crack se creó un hilo donde hablamos sobre el futuro de la humanidad. Unos, pesimistas, argumentaban que el hombre acabaría destruyendo su entorno y a sí mismos… otros, optimistas, creían que en el ser humano sería lo suficientemente inteligente como para superar las adversidades de la evolución.

Yo no conozco el futuro, ignoro lo que le depara el futuro a la humanidad, pero SI SE QUE EL FONDO DEL SER HUMANO ES BUENO.

¿Argumento? Muy simple, la realidad. En Marzo pude ver como unos hombres y mujeres que acababan de salvarse de la muerte, volvían al peligro para salvar a otros hombres y mujeres que ni tan siquiera conocían. Y pude ver como otros hombres y mujeres que podrían haberse quedado a salvo en sus casas, optaron por enfrentarse al peligro para ayudar a sus semejantes.

Tengo la certeza de que el hombre, en su interior más profundo, es bueno. Eso me da esperanza y fuerzas para seguir adelante, fuerza para creer que unos pocos jamás podrán doblegar a la mayoría… y la mayoría es buena, la mayoría somos héroes, la mayoría sacamos de nuestro interior lo mejor que tenemos en los momentos más difíciles.

No olvidemos nunca a quienes han dejado de vivir para regalarnos la vida al resto.

GRACIAS...

INDICE

INDICE DE ANUNCIANTES

CURSO DE PHP
MANEJO DE FORMULARIOS WEB

- FORMULARIOS EN PHP: Trabajaremos con los datos.
- Enviaremos datos de una página a otra.
- Los recogeremos y finalmente los trataremos.

En la navegación web intervienen los siguientes elementos:

- Un Cliente (Navegador Web)
- Un Servidor Web
- Una Petición (Request)
- Una Respuesta (Response)

En pocas palabras, un cliente (Navegador Web) realiza peticiones (Request) a un Servidor Web que responde (Response) las páginas web que están almacenadas en él.

Normalmente cuando se desarrollan páginas Web, es necesario que los datos recogidos en un primer momento se "arrastren" entre las diferentes páginas mientras nos sean necesarios.

Un ejemplo muy claro de esto es un sistema de registro de usuarios, todos nos hemos registrado en algún portal o algún "site", y para realizar el registro completo hay que realizarlo por pasos, donde cada paso es una página Web diferente y los datos obtenidos en la primera página se deben mantener hasta la finalización del registro.

Variables a través de la URL

Para pasar las variables de una página PHP a otra, lo podemos hacer introduciendo dicha variable dentro del enlace hipertexto de la página destino.

La sintaxis sería la siguiente:

```
<a href="destino.php?var1=valor1&var2=valor2&...">enlace</a>
```

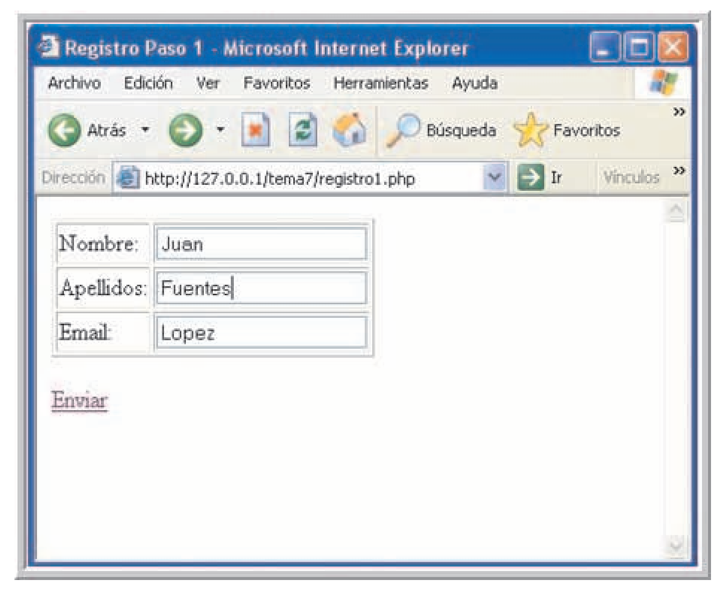
Si nos fijamos, veremos que las variables no poseen el símbolo $ delante. Esto se debe a que en realidad este modo de pasar variables es común a todos los lenguajes (ASP, PHP, JSP,...)

Las variables las recibe la página destino del enlace, y ya podemos trabajar directamente con ellas; pero no siempre se definen automáticamente las variables recibidas por parámetro en las páginas Web, depende de una variable de configuración de PHP: **register_globals**, que tiene que estar activada para que así sea.

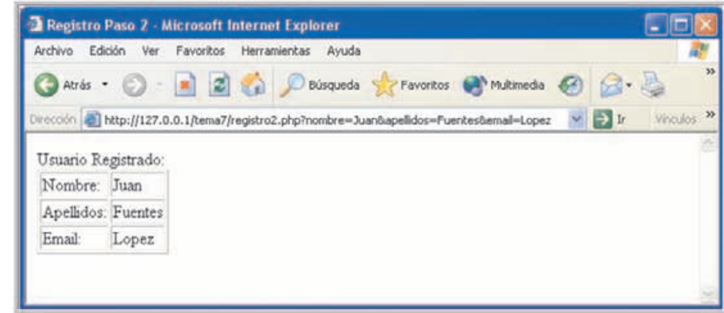
A continuación vamos a explicar esto en un ejemplo. Tenemos dos páginas, registro1.php y registro2.php, que siguen los pasos para un registro simple de usuarios.

```
<HTML>
<HEAD>
<TITLE>Registro Paso 1</TITLE>
</HEAD>
<BODY>
 <FORM NAME="formRegistro1">
 <TABLE BORDER="1">
  <TR>
   <TD>Nombre:</TD>
   <TD><INPUT TYPE="TEXT" NAME="NOMBRE" VALUE=""></TD>
  </TR>
  <TR>
   <TD>Apellidos:</TD>
   <TD><INPUT TYPE="TEXT" NAME="APELLIDOS" VALUE=""></TD>
  </TR>
  <TR>
   <TD>Email:</TD>
   <TD><INPUT TYPE="TEXT" NAME="EMAIL" VALUE=""></TD>
  </TR>
```

```
  </TABLE>
  </FORM>
    <A HREF="javascript:window.location.ref. = 'registro2.php?nombre='
+ document.formRegistro1.NOMBRE.value + '&apellidos=' +
document.formRegistro1.APELLIDOS.value + '&email=' +
document.formRegistro1.EMAIL.value">Enviar</A>
</BODY>
</HTML>
```




```
<HTML>
<HEAD>
<TITLE>Registro Paso 2</TITLE>
</HEAD>
<BODY>
  Usuario Registrado:
  <TABLE BORDER="1">
    <TR>
      <TD>Nombre:</TD>
      <TD><?= $nombre;?></TD>
    </TR>
    <TR>
      <TD>Apellidos:</TD>
      <TD><?= $apellidos;?></TD>
    </TR>
    <TR>
      <TD>Email:</TD>
      <TD><?= $email;?></TD>
    </TR>
  </TABLE>
</BODY>
</HTML>
```



Si nos fijamos en la barra de dirección del navegador de la imagen anterior, en la URL de la página aparecen las variables junto al valor que les hemos rellenado en el formulario.

$HTTP_GET_VARs

También es posible recopilar en una variable tipo array el conjunto de variables que han sido enviadas al script por este método a partir de la variable de sistema $HTTP_GET_VARS, que es un array asociativo.

```
<HTML>
<HEAD>
<TITLE>Registro Paso 2</TITLE>
</HEAD>
<BODY>
  Usuario Registrado:
  <TABLE BORDER="1">
    <TR>
      <TD>Nombre:</TD>
      <TD><? print($HTTP_GET_VARS["nombre"]);?></TD>
    </TR>
    <TR>
      <TD>Apellidos:</TD>
      <TD><? print($HTTP_GET_VARS["apellidos"]);?></TD>
    </TR>
    <TR>
      <TD>Email:</TD>
      <TD><? print($HTTP_GET_VARS["email"]);?></TD>
    </TR>
  </TABLE>
</BODY>
</HTML>
```

Aunque podamos recoger variables que se definen directamente en nuestra página, resulta más seguro utilizar $HTTP_GET_VARS por dos razones:

- La primera es que así nos aseguramos que esa variable viene realmente de la URL
- La segunda, que así nuestro código será más claro cuando lo volvamos a leer, porque quedará especificado que esa variable estamos recibiéndola por la URL.

$_GET

A partir de la versión 4.1.0 de PHP se ha introducido el array asociativo $_GET, que es idéntico a $HTTP_GET_VARS, aunque un poco más corto de escribir.

Variables a través de Formularios

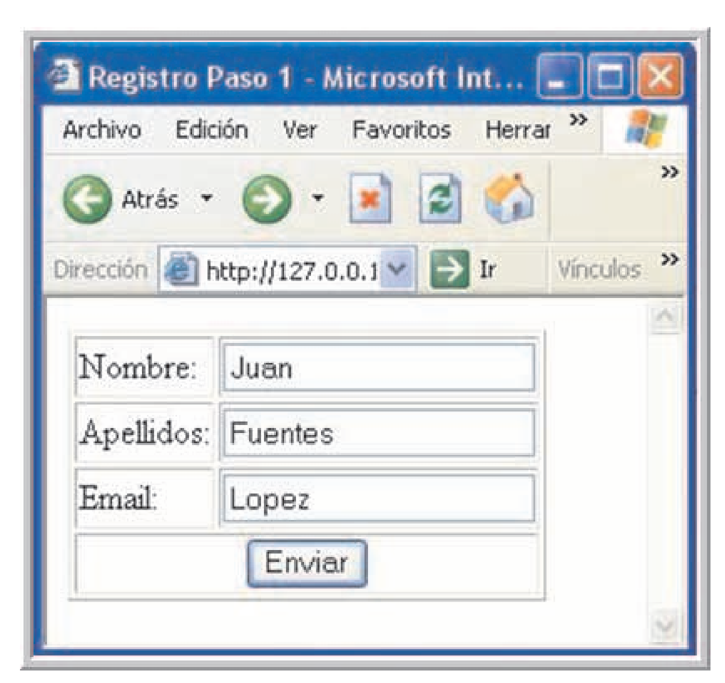
El proceso es similar al explicado para las URLs, pero en esta ocasión el envió de variables de una página PHP a otra no se realiza a través de la URL, si no enviándolas a través de un formulario.

Al igual que en las variables a través de la URL, las variables que se procesan a través del envío de formularios no siempre se definen automáticamente, ello depende de una variable de configuración de PHP: **register_globals**, que tiene que estar activada para conseguir que las variables se definan automáticamente.

Para ver el funcionamiento del procesamiento de variable a través de los formularios vamos a realizar el mismo ejemplo que en el apartado anterior.

Tenemos dos páginas, registro1.php y registro2.php, que siguen los pasos para un registro simple de usuarios.

```
<HTML>
<HEAD>
<TITLE>Registro Paso 1</TITLE>
</HEAD>
<BODY>
  <FORM NAME="formRegistro1" ACTION="registro2.php"
METHOD="POST" >
  <TABLE BORDER="1">
   <TR>
     <TD>Nombre:</TD>
     <TD><INPUT TYPE="TEXT" NAME="NOMBRE" VALUE=""></TD>
   </TR>
   <TR>
     <TD>Apellidos:</TD>
     <TD><INPUT TYPE="TEXT" NAME="APELLIDOS" VALUE=""></TD>
   </TR>
   <TR>
     <TD>Email:</TD>
     <TD><INPUT TYPE="TEXT" NAME="EMAIL" VALUE=""></TD>
   </TR>
   <TR>
     <TD COLSPAN="2" ALIGN="CENTER">
     <INPUT TYPE="SUBMIT" NAME="enviar" VALUE="Enviar">
     </TD>
   </TR>
  </TABLE>
  </FORM>
</BODY>
</HTML>
```



Como podemos ver, el código es exactamente igual al del ejemplo de procesamiento de variables a través de la URL, en lo único que se diferencia es en la línea de definición del formulario:

```
<FORM NAME="formRegistro1" ACTION="registro2.php" METHOD="POST"
>
```

y en el botón que realiza el envío:

```
<INPUT TYPE="SUBMIT" NAME="enviar" VALUE="Enviar">
```

Vamos a prestar especial atención a la línea en la que se define el formulario, ACTION nos indica la página destino y METHOD, nos indica el método (GET o POST) utilizado para el envío del formulario.

Si METHOD es igual a GET las variables del formulario se envían a través de la URL, y si METHOD es igual a POST, el envío de variables no es visible a través de la URL.

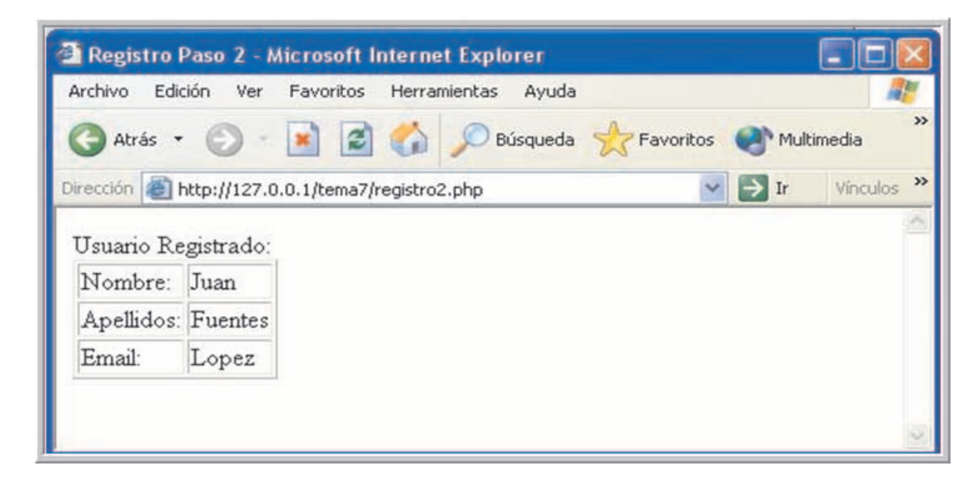
¿Cuál es el mejor a utilizar? Pues esto dependerá del caso concreto en el que nos encontremos, pero siempre queda mucho más elegante y seguro que las variables no estén visibles en la URL. Además la longitud de cadena permitida en la URL es de 255, así que es aconsejable si se van a rellenar muchos datos que en METHOD utilizado sea POST.

```
<HTML>
<HEAD>
<TITLE>Registro Paso 2</TITLE>
</HEAD>
<BODY>
 Usuario Registrado:
 <TABLE BORDER="1">
  <TR>
    <TD>Nombre:</TD>
    <TD><?= $nombre;?></TD>
  </TR>
  <TR>
    <TD>Apellidos:</TD>
    <TD><?= $apellidos;?></TD>
  </TR>
  <TR>
    <TD>Email:</TD>
    <TD><?= $email;?></TD>
  </TR>
 </TABLE>
</BODY>
</HTML>
```

Si ejecutamos el código anterior no recibiremos las variables en registro2.php ¿porqué? PHP distingue entre mayúsculas y minúsculas, es CASE-SENSITIVE, por lo que las variables $nombre, $apellidos y $email son nuevas para el y no tienen ningún valor.

El siguiente código es el correcto:

```
<HTML>
<HEAD>
<TITLE>Registro Paso 2</TITLE>
</HEAD>
<BODY>
 Usuario Registrado:
 <TABLE BORDER="1">
  <TR>
    <TD>Nombre:</TD>
    <TD><?= $NOMBRE;?></TD>
  </TR>
  <TR>
    <TD>Apellidos:</TD>
    <TD><?= $APELLIDOS;?></TD>
  </TR>
  <TR>
    <TD>Email:</TD>
    <TD><?= $EMAIL;?></TD>
  </TR>
 </TABLE>
</BODY>
</HTML>
```



Como podemos observar en la imagen anterior, las variables no están visibles en la URL de la página, pero esto cambiaría si el METHOD utilizado fuese GET.

$HTTP_POST_VARS

También es posible recopilar en una variable tipo array el conjunto de variables que han sido enviadas al script por este método a partir de la variable de sistema $HTTP_POST_VARS, que es un array asociativo.

La sintaxis es la siguiente:

$HTTP_POST_VARS["nombre"]

Aunque podamos recoger variables que se definen directamente en nuestra página, resulta más seguro utilizar $HTTP_POST_VARS por dos razones, la primera que así nos aseguramos que esa variable viene realmente de la URL y la segunda, que así nuestro códigoserá más claro cuando lo volvamos a leer, porque quedará especificado que esa variable estamos recibiéndola por la URL

$_POST

A partir de PHP 4.1.0 se pueden recoger las variables de formulario utilizando también el array asociativo $_POST, que es el mismo que $HTTP_POST_VARS, pero más corto de escribir.

En el próximo número...

En el próximo número comenzaremos a programar sockets, programar socket nos permitirá hacer un emulador de Telnet, hacer un FTP, ... y todo gracias al PHP ;)

SERVIDOR DE HXC MODO DE EMPLEO

- Hack x Crack ha habilitado tres servidores para que puedas realizar las prácticas de hacking.

- Las IPs de los servidores de hacking las encontrarás en EL FORO de la revista (www.hackxcrack.com). Una vez en el foro entra en la zona COMUNICADOS DE HACK X CRACK (arriba del todo) y verás varios comunicados relacionados con los servidores. No ponemos las IP aquí porque es bueno acostumbrarte a entrar en el foro y leer los comunicados. Si hay alguna incidencia o cambio de IP o lo que sea, se comunicará en EL FORO.

- Actualmente tienen el BUG del Code / Decode. La forma de "explotar" este bug la explicamos extensamente en los números 2 y 3. Lo dejaremos así por un tiempo (bastante tiempo ;) Nuestra intención es ir habilitando servidores a medida que os enseñemos distintos tipos de Hack.

- En los Servidores corre el Windows 2000 con el IIS de Servidor Web. No hemos parcheado ningún bug, ni tan siquiera el RPC y por supuesto tampoco hemos instalado ningún Service Pack. Para quien piense que eso es un error (lógico si tenemos en cuenta que el RPC provoca una caída completa del sistema), solo decirte que AZIMUT ha configurado un firewall desde cero que evita el bug del RPC, (bloqueo de los puertos 135 (tcp/udp), 137 (udp), 138 (udp), 445 (tcp), 593 (tcp)). La intención de todo esto es, precisamente, que puedas practicar tanto con el CODE/DECODE como con cualquier otro "bug" que conozcas (y hay cientos!!!). Poco a poco iremos cambiando la configuración en función de la experiencia, la idea es tener los Servidores lo menos parcheados posibles pero mantenerlos operativos las 24 horas del día. Por todo ello y debido a posibles cambios de configuración, no olvides visitar el foro (Zona Comunicados) antes de "penetrar" en nuestros servidores.

- Cada Servidor tiene dos unidades (discos duros duros):
* La unidad c: --> Con 40GB y Raíz del Sistema
* La unidad d: --> Con 40GB
* La unidad e: --> CD-ROM

Nota: Raíz del Servidor, significa que el Windows Advanced Server está instalado en esa unidad (la unidad c:) y concretamente en el directorio por defecto \winnt\ Por lo tanto, la raíz del sistema está en c:\winnt\

- El IIS, Internet Information Server, es el Servidor de páginas Web y tiene su raíz en c:\inetpub (el directorio por defecto)

Nota: Para quien nunca ha tenido instalado el IIS, le será extraño tanto el nombre de esta carpeta (c:\inetpub) como su contenido. Pero bueno, un día de estos os enseñaremos a instalar vuestro propio Servidor Web (IIS) y detallaremos su funcionamiento.

De momento, lo único que hay que saber es que cuando TÚ pongas nuestra IP (la IP de uno de nuestros servidores) en tu navegador (el Internet explorer por ejemplo), lo que estás haciendo realmente es ir al directorio c:\Inetpub\wwwroot\ y leer un archivo llamado default.htm.

Nota: Como curiosidad, te diremos que APACHE es otro Servidor de páginas Web (seguro que has oído hablar de él). Si tuviésemos instalado el apache, cuando pusieses nuestra IP en TU navegador, accederías a un directorio raíz del Apache (donde se hubiese instalado) e intentarías leer una página llamada index.html ... pero... ¿qué te estoy contando?... si has seguido nuestra revista ya dominas de sobras el APACHE ;)

Explicamos esto porque la mayoría, seguro que piensa en un Servidor Web como en algo extraño que no saben ni donde está ni como se accede. Bueno, pues ya sabes dónde se encuentran la mayoría de IIS (en \Inetpub\) y cuál es la página por defecto (\Inetpub\wwwroot\default.htm). Y ahora, piensa un poco… … ¿Cuál es uno de los objetivos de un hacker que quiere decirle al mundo que ha hackeado una Web? Pues está claro, el objetivo es cambiar (o sustituir) el archivo default.html por uno propio donde diga "hola, soy DIOS y he hackeado esta Web" (eso si es un lamer ;)

A partir de ese momento, cualquiera que acceda a ese servidor, verá el default.htm modificado para vergüenza del "site" hackeado. Esto es muy genérico pero os dará una idea de cómo funciona esto de hackear Webs ;)

- Cuando accedas a nuestro servidor mediante el CODE / DECODE BUG, crea un directorio con tu nombre (el que mas te guste, no nos des tu DNI) en la unidad d: a ser posible y a partir de ahora utiliza ese directorio para hacer tus prácticas. Ya sabes, subirnos programitas y practicar con ellos :) ... ... ¿cómo? ¿que no sabes crear directorios mediante el CODE/DECODE BUG... repasa los números 2 y tres de Hack x Crack ;p

Puedes crearte tu directorio donde quieras, no es necesario que sea en d:\mellamojuan. Tienes total libertad!!! Una idea es crearlo, por ejemplo, en d:\xxx\system32\default\10019901\mellamojuan (ya irás aprendiendo que cuanto mas oculto mejor :)

Es posiblemente la primera vez que tienes la oportunidad de investigar en un servidor como este sin cometer un delito (nosotros te dejamos y por lo tanto nadie te perseguirá). Aprovecha la oportunidad!!! e investiga mientras dure esta iniciativa (esperemos que muchos años).

- En este momento tenemos mas de 600 carpetas de peña que, como tu, está practicando. Así que haznos caso y crea tu propia carpeta donde trabajar.



MUY IMPORTANTE...

MUY IMPORTANTE!!!!! Por favor, no borres archivos del Servidor si no sabes exactamente lo que estás haciendo ni borres las carpetas de los demás usuarios. Si haces eso, lo único que consigues es que tengamos que reparar el sistema servidor y, mientras tanto, ni tu ni nadie puede disfrutar de él :(Es una tontería intentar "romper" el Servidor, lo hemos puesto para que disfrute todo el mundo sin correr riesgos, para que todo el mundo pueda crearse su carpeta y practicar nuestros ejercicios. En el Servidor no hay ni Warez, ni Programas, ni claves, ni nada de nada que "robar", es un servidor limpio para TI, por lo tanto cuídalo un poquito y montaremos muchos más :)

SERIE XBOX LIFE (IV)
CONVIRTIENDO NUESTRA XBOX EN UN CENTRO MULTIMEDIA.

- ¿Podemos ver películas en DivX con la XBOX?
- ¿Y escuchar musica en formato MP3?
- ¿Y visualizar fotos?

SI, SI y SI

Aquí estamos un mes más, antes de todo quiero agradecer a los que apoyan esta serie, es muy grato ver recompensado el esfuerzo. GRACIAS.

Este mes vamos a descubrir los distintos lectores DVD que tiene la consola Xbox y vamos a instalar y configurar unos programas que nos permitan ver películas DivX, escuchar MP3, ver fotos, incluso ahorrarnos el tener que comprar el Kit DVD para ver las películas en este formato.

Como es de costumbre en esta serie, os voy a detallar lo que necesitamos para hacer todas estas cosas.

Lo que necesitamos:

- Xbox con mod chip.
- Evox instalado en nuestra consola
- Media Center (la versión más reciente)
- DVDX2.
- ZXBTOOLS (Ultima versión).
- Pc Preparado.

Todos los programas los podréis encontrar en la Web de la revista (www.hackxcrack.com).

Los DVD de XBOX:

Algunos de vosotros sabréis ya que hay 3 fabricantes distintos de lectores para esta consola, para quien no lo sepa, las marcas son Samsung, Thompson y Philips.

Os podéis estar preguntando, ¿para qué os cuento esto?

Pues muy sencillo, dependiendo del lector que tengamos, podremos usar CD´s, CD-RW, DVD y determinadas marcas de cada uno de éstos.

Samsung:



Este lector es el mejor, lee lo que le metas, tanto CD´s como DVD´s y prácticamente de cualquier marca. En la imagen de arriba podéis ver cómo es la bandeja de este lector, así podréis saber qué lector lleva vuestra consola sin necesidad de abrirla.

Philips:



Este lector es el segundo si los ordenamos de mejor a peor. Lee algunos CD´s y DVD´s de marcas conocidas, siempre y cuando se calibre la lente para que los lea, ya que el láser encargado de leer los CD´s está tan bajo de potencia que no es capaz de leerlos.

El calibrar la lente lleva sus riesgos, ya que le acortas la vida y en muchos casos, si se hace mal, puede dejar de funcionar.

Thompson:



Este podríamos decir que es el peor, ya que no lee CD´s, solo CD-RW y de lagunas marcas como VERBATIM.

Con los DVD también tenemos el mismo problema y lo más aconsejable es usar los de marca PRIMCO.

Instalando DVDX2:

Nos lo descargamos de la Web y lo descomprimimos en una carpeta del pc. Veremos dos archivos: el dvdx_v2.0.iso y dvdx2.txt.

El txt lo podemos borrar, sólo trae información de quien lo creó.

Extraemos dvdx_v2.0.iso con el ZXBTOOLS (como os enseñé el mes pasado) y tendremos una carpeta llamada "Media" y un archivo llamado "default.xbe".

Ahora tenemos que pasar la carpeta y el archivo al HD de la consola. Si tenéis configurado el evox como expliqué anteriormente, copiáis todo a la carpeta MP que creamos en el artículo de Evolution X. Si no fuera así, lo copiáis a la carpeta que corresponda.

La transferencia la hacemos por FTP. Copiamos todo, reiniciamos la consola y lo veremos en el apartado donde los hayáis copiado (en mi caso Media Players).

Nos preparamos unas palomitas, arrancamos el DVDX2, metemos la película y a disfrutar. Con esto tendréis que usar el mando de la consola para manejaros por los menús, pero os habréis ahorrado 30€.

Instalando y Configurando El Xbox Media Center:

Lo descargamos de la web, lo descomprimimos en una carpeta y veremos esto:



Nos metemos en la carpeta "xbmc-2004-01-02" y veremos esto:



Editamos con el bloc de notas el archivo "XboxMediaCenter.xml". Aquí os digo os enseño cómo he configurado el archivo, lo configuramos a nuestro gusto (lo que está en negrita es la explicación):

<xboxmediacenter>

#Skin que cargará al iniciar el programa, lo dejamos como está!#

<skin>MediaCenter</skin>

#Dirección IP, Máscara subred, Puerta de enlace por defecto, Nombre del server (Dejad en blanco si queréis usar los valores del Evolution X) !#

<ipadres>-</ipadres>
<netmask>-</netmask>
<defaultgateway>-</defaultgateway>

```
  <nameserver>192.168.0.1</nameserver>
<CDDBIpAdres>194.97.4.18</CDDBIpAdres>
<CDDBEnabled>yes</CDDBEnabled>
```

#Determina las unidades del hd de la consola que se puede usar !#

```
  <useFDrive>yes</useFDrive>
  <useGDrive>no</useGDrive>
```

#Valores del servidor proxy!#

```
  <httpproxy>-</httpproxy>
  <httpproxyport>-</httpproxyport>
```

#Servidor remoto que nos dará la fecha y hora!#

```
<timeserver>207.46.248.43</timeserver>
```

#El dashboard que arrancará al salir!#

```
<dashboard>C:\evoxdash.xbe</dashboard>
```

#Directorios alternativos para los subtítulos de las películas, no os olvidéis de crear esta carpeta !#

```
<subtitles>$HOME\subtitles</subtitles>
```

#Lo dejamos como está !#

```
  <startwindow>0</startwindow>
```

#Extensiones de las fots, Videos y Sonidos que queremos que acepte. Si falta alguna, agregadla!#

```
<pictureextensions>.png|.jpg|.jpeg|.bmp|.gif|.ico|.tif|.ti
ff|.tga|.pcx</pictureextensions>

<musicextensions>.m4a|.flac|.ac3|.aac|.nfo|.pls|.rm|.
mpa|.wav|.wma|.ogg|.mp3|.mp2|.m3u</musicextensions>

<videoextensions>.nfo|.rm|.m3u|.ifo|.mov|.qt|.divx|.xv
id|.bivx|.vob|.pva|.wmv|.asf|.asx|.ogm|.m2v|.avi|.bin|.
dat|.mpg|.mpeg|.mkv|.avc|.vp3|.svq3|.nuv|.viv|.dv|.fli
</videoextensions>

  <!-- path where xbmc should store its thumbnails !#

<thumbnails>$HOME\thumbs</thumbnails>
```

#Directorio donde almacenará los atajos (lo dejamos como está !#

```
<shortcuts>$HOME\shortcuts</shortcuts>
```

#Ruta donde guarda el contenido destinado a la memoria caché (lo dejamos como está) !#

```
  <imdb>$HOME\imdb</imdb>
```

#Ruta donde guarda el contenido destinado a la memoria caché de los álbumes de fotos (lo dejamos como está) !#

```
  <albums>$HOME\albums</albums>
```

#Lo dejemos como esta, son los valores del mando !#

```
  <delays>
    <remote>
      <move>220</move>
      <repeat>300</repeat>
    </remote>
    <controller>
      <move>220</move>
      <repeat>220</repeat>
    </controller>
  </delays>
```

Aquí podemos especificar los iconos que se mostrarán en cada apartado !#
#Para los archivos !#

```
  <filetypeicons>
    <xbe>defaultProgram.png</xbe>
  </filetypeicons>
```

#Esta es la configuración base del menú, es decir lo que vamos a ver en el programa, en cada sección. Podemos agregar o quitar la que queramos, en name ponemos el nombre que queramos y en path ponemos la ruta a la carpeta, por ejemplo, he quitado el acceso a c: y os aconsejo que hagáis lo mismo.!#

#Configuración para Mis programas#

```
 <myprograms>
    <bookmark>
      <name>Programas en E:</name>
      <path>E:\apps\</path>
    </bookmark>
    <bookmark>
      <name> Programas en F:</name>
      <path>F:\apps\</path>
    </bookmark>
    <bookmark>
      <name>Programas en DVD </name>
      <path>D:\</path>
    </bookmark>
 </myprograms>
```

#Configuración para las fotos !#

```
<pictures>
  <default></default>
 <bookmark>
    <name>Fotos E </name>
    <path>E:\fotos\</path>
  </bookmark>
  <bookmark>
    <name>Fotos F:</name>
    <path>F:\fotos\</path>
  </bookmark>
  <bookmark>
    <name>Fotos en DVD</name>
    <path>D:\</path>
  </bookmark>
```

#Ahora llega lo interesante, estos apartado son para hacer stemig o lo que es lo mismo, pillar los archivos en este caso las fotos del PC, por medio del cable RJ-45, y aquí se pueden definir los valores. Podéis quitar los que no uséis; yo solo uso el SMB, que más adelante os explicaré cómo se configura. A partir de aquí quitaré los demás !#

```
  <bookmark>
    <name>XNS</name>
    <path>xns://192.168.0.1:1400/</path>
  </bookmark>
  <bookmark>
    <name>XBMS</name>
    <path>xbms://192.168.0.1:1400/</path>
  </bookmark>
```

#Aquí está la configuración del strem por SMB, dependiendo de cómo hayáis compartido la carpeta en el pc, se configura de una manara o de otra!#

```
<bookmark>
    <name>SMB</name>
```

#Este es un ejemplo smb://dominio;Usuario:contraseña@servidor/carpeta compartida !#

#Así tengo configurado el mío !#

```
<path>smb://192.168.0.1/fotos/</path>
</bookmark>
  </pictures>
```

#Mis archivos !#

#Más de lo mismo, venga, a configurar !#

```
<files>
   <bookmark>
      <name>E :</name>
      <path>E:\</path>
    </bookmark>
    <bookmark>
      <name>F: drive</name>
      <path>F:\</path>
    </bookmark>
<bookmark>
      <name>DVD</name>
      <path>D:\</path>
    </bookmark>
    <bookmark>
      <name>SMB</name>
            <path>smb://192.168.0.1/archivos/</path>
    </bookmark>
  </files>
```

Mi música !#

```
  <music>
    <default></default>
   <bookmark>
      <name>Musica E: </name>
      <path>E:\MP3\</path>
    </bookmark>
    <bookmark>
      <name>Musica F:</name>
      <path>F:\MP3\</path>
    </bookmark>
    <bookmark>
      <name>MP3 DVD</name>
      <path>D:\</path>
    </bookmark>
    <bookmark>
      <name>SMB</name>
         <path>smb://192.168.0.1/mp3/</path>
</bookmark>
  </music>
```

```
#Mis videos !#

 <video>
   <default></default>
<bookmark>
     <name>Viedos E:</name>
     <path>E:\videos\</path>
   </bookmark>
   <bookmark>
     <name>Videos F:</name>
     <path>F:\videos\</path>
   </bookmark>
   <bookmark>
     <name>Videos DVD</name>
     <path>D:\</path>
   </bookmark>
     <name>SMB</name>
<path>smb://192.168.0.1/videos/</path>    </bookmark>
 </video>
</xboxmediacenter>
```

Ya se ha acabado la configuración, así que guardamos los cambios, y salimos del bloc de notas.

Ahora tenemos que compartir las carpetas en nuestro PC para que a través de la red podamos cargar películas o escuchar
música.

En Windows XP se haría de la siguiente manera:

Abrimos el explorador de Windows, nos situamos encima de la carpeta que queremos compartir, por ejemplo el de videos, pinchamos con el botón derecho sobre la carpeta y pinchamos en propiedades:



Pinchamos en la pestaña compartir, seleccionamos la casilla "Compartir esta carpeta en la red", cambiamos el nombre si ya esta ocupado y aceptamos.

Esto lo hacemos con todas las carpetas que hemos puesto en los apartados de SMB.

Pues ya está todo. Pasamos todo lo que hay en esta carpeta a su carpeta correspondiente del HD de la consola, lo arrancamos y a disfrutar.

Con un poco de investigación os haréis con el control de todo, es muy sencillo e intuitivo.

El mes que viene jugaremos online GRATIS. Hasta entonces.

Salu2.

PROGRAMACION EN GNU LINUX

MEMORIA COMPARTIDA

el_chaman. Luis U. Rodríguez Paniagua

- Intercambio de datos entre procesos
- Memoria compartida
- Colas de mensajes
- pipes
- sockets

1. Presentación

En el presente artículo veremos como es posible intercambiar datos entre dos procesos. Dispondremos de tres mecanismos básicos: Memoria compartida, colas de mensajes y empleo de *pipes* y *sockets*. Analizaremos en el presente artículo las dos primeras formas.

2. Memoria compartida.

Este será el método más rápido de comunicar dos procesos entre sí. Consiste en hacer que dos procesos compartan una zona común de memoria de manera que los dos procesos podrán leer y escribir en dicha zona de memoria.

Esto, que en teoría es sencillo de entender, a la hora de llevarlo a la práctica se complica un poco. Cada vez que ejecutamos un proceso, este tiene asociada una parte de la memoria exclusivamente para él. Ahí será donde guarde sus datos (variables, pila, etc...).

Si en alguna ocasión este proceso intenta acceder a una zona de memoria a la que no tiene acceso u otro proceso intenta acceder a la memoria reservada para este proceso, se estará produciendo una *violación de segmenteto* con lo cual el programa fallará.

El encargado de controlar esto es el S.O., y afortunadamente en nuestro caso, este dispone de la interfaz adecuada que nos permitirá crear zonas de memoria especiales que podrán ser compartidas entre varios procesos.

2.1. Solicitud de memoria compartida

La primera operación que debemos aprender es la necesaria para solicitar una zona de memoria compartida. La manera de hacer esto es mediante la función:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int shmflg);
```

Retorna: -1 en caso de error, o el identificador de la zona de memoria compartida en otro caso.

shmget reserva una zona de memoria de tamaño size *bytes* devolviendo un identificador que nos permitirá manejar dicha zona de memoria. El objetivo de key es similar al visto en la llamada de semáforos semget (es decir, pasar una clave única que servirá en nuestro caso para identificar de manera única la zona de memoria asociada a nuestro proceso).

En cuanto a shmflag será una máscara de bits que indicarán el modo de adquisición del identificador devuelto por shmget:

- Si el bit IPC_CREAT está activo, la zona compartida de memoria se creará a no ser que haya sido creada por otros procesos.
- Si los bits IPC_CREAT e IPC_EXCL están activos simultáneamente, la llamada a shmget falla en caso de que la memoria compartida hubiese sido reservada previamente.
- Además de estos valores podremos indicar los permisos: 0400 (lectura para el usuario), 0200 (modificación para el usuario), 0060

(permiso de lectura y modificación para el grupo) y 0006 (permiso de lectura y modificación para el resto de los usuarios).

Para crear una zona de memoria compartida de 1KByte (1024 bytes) con permisos de escritura lectura sólo para usuario teclearíamos:

```
int shmid;
    ...
    if (( shmid = shmget(IPC_PRIVATE, 1024, IPC_CREAT | 0600 )) == -1)
    {
        fprintf(stderr, " ERROR al reservar memoria compartida\n");
    }
    ....
```

2.2. Control de memoria compartida

Una vez que hemos reservado una zona de memoria compartida, podemos realizar diversas operaciones de control sobre dicha zona con el fin de establecer distintos atributos de la misma. Estas operaciones de control las realizará la función shmctl.

La declaración de esta función es:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Retorna 0 en caso de éxito y -1 en caso de error.

shmid es el identificador de memoria compartida sobre el que vamos a operar.

cmd es la operación de control que queremos realizar. Este parámetro puede tomar los siguientes valores:

- IPC_STAT Lee el estado de la estructura de control de la memoria y lo devuelve a través de la zona de memoria apuntada por *buf. La estructura de esta zona de memoria es:

```
struct shmid_ds {
        struct ipc_perm shm_perm;  /* Estructura de permisos */
        int shm_segsz;   /* Tamaño del área de memoria (bytes) */
        time_t shm_atime;         /* last attach time */
        time_t shm_dtime;         /* last detach time */
        time_t shm_ctime;         /* last change time */
        unsigned short shm_cpid;   /* pid of creator */
        unsigned short shm_lpid;   /* pid of last operator */
        short shm_nattch;         /* no. of current attaches */
        ...
    };

    struct ipc_perm {
        key_t  key;
        ushort uid;   /* owner euid and egid */
        ushort gid;
        ushort cuid;  /* creator euid and egid */
        ushort cgid;
        ushort mode;  /* lower 9 bits of access modes */
        ushort seq;   /* sequence number */
    };
```

- IPC_SET Inicializa algunos de los campos de la estructura de control de la memoria compartida. El valor de estos campos se toma de la estructura referenciada por *buf.
- IPC_RMID Elimina la zona de memoria asociada al identificador shmid. Si esta zona de memoria compartida está siendo usada por varios procesos, no se elimina hasta que todos los procesos liberen esta zona de memoria compartida.
- SHM_LOCK Bloquea en memoria el segmento identificado por shmid impidiendo el intercambio sobre él. Esta operación sólo puede ser realizada por el superusuario.
- SHM_UNLOCK. Desbloquea en memoria el segmento identificado por shmid. Esta operación sólo la puede realizar el superusuario.

2.3. Operaciones con memoria compartida

Para poder usar una zona de memoria compartida que previamente hemos creado, esta debe ligarse al espacio de direcciones de memoria asignadas a nuestro proceso. La forma en la que vamos a poder ligar o desligar una zona de memoria compartida al espacio de direcciones de nuestro proceso es mediante el empleo de las funciones shmat y shmdt.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr, int shmflg);
```

shmat es la función encargada de ligar una zona de memoria compartida al espacio de direcciones de memoria local del proceso.

Retorna: la dirección a la que está unida la zona de memoria compartida. -1 en caso de error.

Los parámetros que recibe son: shmid que será el identificador de la zona de memoria compartida a enlazar. shmaddr será la dirección virtual donde queremos que nuestro proceso "vea" la zona de memoria compartida. Como esto lo debe asignar el núcleo, puede que no siempre se nos devuelva el puntero apuntando a shmaddr, por lo que se suele poner este parámetro a 0 y dejar la responsabilidad al S.O.

En cuanto a shmflg, se trata de una máscara de bits que indica el modo de acceso a memoria. Si queremos hacer un acceso de sólo lectura, indicaremos, por ejemplo, SHM_RDONLY (Más información en la página de manual correspondiente).

La función encargada de desligar la zona de memoria compartida es shmdt:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

Retorna: 0 en caso de éxito, -1 en caso de error.

Esta función recibirá como único parámetro: shmaddr que es el puntero que señala a la zona de memoria compartida a desligar.

2.4. Ejemplo de uso

En este primer ejemplo vamos a hacer que un proceso hijo llene un vector de caracteres con una frase. A continuación el proceso padre, espera a que termine el hijo y lee dicho vector:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

int main(int argc, char *argv[])
{
    int shmid, i, semid, estado, pid;
    struct sembuf operaciones;
    char *mensaje;
    key_t clave;

    //
    // Zona de cración de memoria compartida
    //
    // Creación de la memoria compartida
    // Primero la clave de recurso IPC
    if( (clave = ftok("shm.c",'K'))==-1)
    {
        fprintf(stderr," Error al crear la clave\n");
        return -1;
    }
    // Ahora la memoria compartida. Tamaño: 1024 caracteres
    if( (shmid=shmget(clave, 1024 * sizeof(char), IPC_CREAT | 0600))==-1)
    {
        fprintf(stderr," Error al crear segmento de memoria compartida\n");
        return -1;
    }
    // Ahora debemos enlazar la zona de memoria compartida con la
    // memoria local del proceso.
    if( (mensaje=(char *)shmat(shmid, 0, 0))==-1)
    {
        fprintf(stderr," Error al ligar la memoria compartida\n");
        return -1;
    }

    // Comienza el lanzamiento de procesos
    //
    if((pid=fork())==-1)
    {
        // Escribimos en la memoria compartida
```

```
        fprintf(stderr, "Error al lanzar proceso hijo\n");
        return -1;
    }

    if (pid != 0) // Somos el padre
    {
        // leemos de la memoria compartida
        // Esperamos a que termine el proceso hijo
        wait(&i);
        for(i=0;i<1024;i++)
        printf("%c-", mensaje[i]);
        return 0;
    }
    else // Somos el hijo
    {
        // Escribimos algo
        strncpy(mensaje, "Hola, esto es una prueba de memoria compartida", 1024);
        return 0;
    }

    // Limpieza de memoria compartida
    if((shmdt(mensaje))==-1)
    {
        fprintf(stderr," Error al desaligar memoria compartida\n");
    }
    return 0;
}
```

Pero es en los siguientes ejemplos donde muestra un poco más de potencia el empleo de la memoria compartida. Imaginemos que tenemos dos programas cuyo código es el siguiente:

```
//
// Servidor que muestra lo que escribe el cliente
// Compilar con: gcc shm_s.c -o shm_s
//
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

int main(int argc, char *argv[])
{

    int shmid, i, estado;
    struct sembuf operaciones;
    char *mensaje, buffer[80];
    key_t clave;

    //
    // Zona de creación de memoria compartida
    //
    // Creación de la memoria compartida
    // Primero la clave de recurso IPC
    if( (clave = ftok("shm.c",'K'))==-1)
    {
        fprintf(stderr," Error al crear la clave\n");
        return -1;
    }
    // Ahora la memoria compartida. Tamaño: 1024 caracteres
    if( (shmid=shmget(clave, 1024 * sizeof(char), IPC_CREAT | 0600))==-1)
    {
        fprintf(stderr," Error al crear segmento de memoria compartida\n");
        return -1;
    }
    // Ahora debemos enlazar la zona de memoria compartida
    // con la memoria local del proceso.
    if( (mensaje=(char *)shmat(shmid, 0, 0))==-1)
    {
        fprintf(stderr," Error al ligar la memoria compartida\n");
        return -1;
    }

    // Comienza el lanzamiento de procesos
    //
    printf("\n Lector\n");
    do{
        for(i=0;i<80;i++)
            printf("%c",mensaje[i]);
        printf("\n");
        sleep(1);

    }while(mensaje[0]!='0');
```

```
        // Limpieza de memoria compartida
        if((shmdt(mensaje))==-1)
        {
                fprintf(stderr," Error al desaligar memoria compartida\n");
        }
        return 0;
}
```

El servidor examina el buffer de memoria compartida cada segundo e imprime su contenido.

El cliente será el encargado de llenar dicho buffer:

```
//
// Cliente que escribe lo que mostrará el servidor
// Compilar con: gcc shm_c.c -o shm_c
//
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

int main(int argc, char *argv[])
{
        int shmid, i,j, estado;
        struct sembuf operaciones;
        char *mensaje, buffer[80];
        key_t clave;

        //
        // Zona de cración de memoria compartida
        //
        // Creación de la memoria compartida
        // Primero la clave de recurso IPC
        if( (clave = ftok("shm.c",'K'))==-1)
        {
                fprintf(stderr," Error al crear la clave\n");
                return -1;
        }
        // Ahora la memoria compartida. Tamaño: 1024 caracteres
        if( (shmid=shmget(clave, 1024 * sizeof(char), IPC_CREAT | 0600))==-1)
        {
            fprintf(stderr," Error al crear segmento de memoria compartida\n");
                return -1;
        }
        // Ahora debemos enlazar la zona de memoria compartida con la
        // memoria local del proceso.
        if( (mensaje=(char *)shmat(shmid, 0, 0))==-1)
        {
                fprintf(stderr," Error al ligar la memoria compartida\n");
                return -1;
        }

        // Comienza el lanzamiento de procesos
        //
        printf("\n Escriba 0 para salir\n");
        do{
                printf("\n > ");
                i=0;
                do
                {
                 fscanf(stdin, "%c", &mensaje[i]);
                 i++;
                }while(mensaje[i-1]!=10);
                //scanf("%s", buffer);
                //strncpy(mensaje, buffer, 80);
                for(j=i;j<1024;j++)
                        mensaje[j]=' '; // rellenamos el resto del buffer
                                        // con espacios
        }while(mensaje[0]!='0');

        // Limpieza de memoria compartida
        if((shmdt(mensaje))==-1)
        {
                fprintf(stderr," Error al desaligar memoria compartida\n");
        }
        return 0;
}
```

Una vez compilados estos programas, en dos terminales distintos, abrimos primero un cliente y escribimos algo:

```
luis@leonov:~/articulos/articulo11_el_chaman/code$ ./shm_c
 Escriba 0 para salir
 > hola
```

Y en otro terminal ejecutamos el servidor:

```
luis@leonov:~/articulos/articulo11_el_chaman/code$ ./shm_s
 Lector
hola
```



A partir de este momento cada cosa que escribamos en el cliente será mostrado en el servidor cada segundo.

Estos dos programas aún se pueden mejorar un poco de manera que, por ejemplo, el servidor sólo muestre algo cuando el buffer está lleno (vaciándolo por ejemplo tras recibir el último dato o sincronizando los procesos mediante semáforos). Dejo a los lectores el practicar con estos ejercicios.

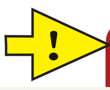
CURSO DE TCP/IP (2ª Entrega)

EL PROTOCOLO DE TRANSPORTE UDP (PROTOCOLO DE DATAGRAMAS DE USUARIO)

- Vamos a descubrir las herramientas necesarias para "crear" paquetes de red
- Empezaremos a desgranar las capas de red
- Trataremos el Protocolo de Datagramas de Usuario

1. INTRODUCCIÓN

Espero que no os perdieseis el primer artículo del curso de TCP/IP (Transmisión Control Protocol / Internet Protocol --- Protocolo de Control de Transmisión / Protocolo de Internet), porque voy a asumir que habéis comprendido a grandes rasgos el concepto de capas de protocolos, y el mecanismo básico de transporte de paquetes de datos a través de Internet (o cualquier otra red TCP/IP).



NOTA DE PC PASO A PASO / HXC

Para todos aquellos que no tienen la primera entrega del curso de TCP / IP publicada en el número 17 de PC PASO A PASO, hemos decidido pasarla a formato PDF y liberarla en la Web de la revista (www.hackxcrack.com).

También aprovechamos esta "nota" para indicar a nuestros lectores que todos los artículos liberados y los programas que mencionamos en los artículos están disponibles en la sección ARTÍCULOS LIBERADOS Y DESCARGAS de la Web.

Para esta primera "lección" he escogido un protocolo muy sencillo, el protocolo UDP (User Datagram Protocol --- Protocolo de Datagramas de Usuario), para así poder extenderme más en explicaros algunas herramientas que nos pueden ser útiles a lo largo de todo el curso. De momento, estas herramientas las utilizaremos para manejar el protocolo UDP, pero en futuras lecciones exprimiremos al máximo las herramientas para hacer maravillas con las demás capas de protocolos: TCP, IP, ARP (Address Resolution Protocol --- Protocolo de Resolución de Direcciones), etc.

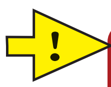
Empezaré explicando a grandes rasgos el mecanismo de transporte de un paquete UDP a través de Internet para luego detallar las cabeceras UDP y, por último, veremos todo esto de forma práctica utilizando unas herramientas que nos permiten construir paquetes UDP desde cero (los llamados "raw sockets").

Pero, antes de empezar, he de aclararos un asunto. Supongo que algunos de vosotros os preguntareis qué ha pasado con la serie RAW. Mis intenciones no eran de ninguna manera sustituir la serie RAW por el curso de TCP/IP, si no desarrollar ambos en paralelo. Pero es increíble de qué manera pueden conspirar juntas las circunstancias de la vida para cambiar por completo tu rumbo cuando menos te lo esperas. En poco más de un mes me han surgido una serie de problemas de todo tipo (familiares, personales, de salud, laborales, y académicos) por los cuales ahora (y sin plazo definido) dispongo de muchísimo menos tiempo.

He barajado un poco mis posibilidades, y creo que lo que puedo permitirme de momento es continuar sólo con el curso de TCP/IP, aunque no descarto publicar algún otro artículo de la serie RAW algún mes que consiga juntar algo más de tiempo. Con esto lo que os quiero

decir es que no os prometo nada con respecto a la serie RAW, pero que haré lo que esté en mi mano para que no termine aquí, aunque haya que avanzar ahora mucho más despacio.

Gracias por vuestra comprensión. ;-)



NOTA EDITORIAL

Para los nuevos lectores, simplemente decir que la serie de artículos denominada "SERIE RAW" ha estado presente en casi todos los números de esta publicación y ha constituido uno de los pilares de la misma.

La Serie RAW ya nos ha aportado conocimientos sobre los Protocolos más importantes y desde hace mucho tiempo se nos ha reclamado el inicio de un curso de TCP / IP. Finalmente, desde el anterior número 17 por fin ese curso ha llegado.

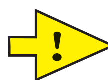
Tal y como nos describe el autor, esperamos poder hacer más entregas de la Serie Raw en futuros números, pero de forma más dilatada en el tiempo. Mientras tanto disfrutemos de este excelente curso de TCP / IP, una serie de artículos ampliamente reclamada y esperada por nuestros lectores.

2. FUNCIONAMIENTO DEL PROTOCOLO UDP

Los que os perdisteis la introducción de este curso no tenéis que preocuparos, ya que este primer punto resume en cierto modo lo que expliqué entonces. Aún así, os recomiendo que leáis la introducción, ya que aquí voy a resumir en un par de páginas lo que allí explicaba en más de 20 páginas.

Para los que sí que leísteis la introducción, este punto creo que os puede ayudar a aclarar las ideas a modo de resumen.

A lo largo de la serie RAW os expliqué varios protocolos que funcionaban sobre TCP, pero sólo uno que funcionase sobre UDP: el protocolo DNS. Vamos a utilizar este protocolo como ejemplo para ver qué ocurre desde que un cliente solicita una consulta DNS a un servidor, hasta que éste cliente recibe la respuesta pertinente. Os aconsejo que leáis el artículo sobre DNS de la serie RAW, que se encuentra en el número 14 de la revista, aunque voy a empezar haciendo una introducción sobre este protocolo.



NOTA DE PC PASO A PASO / HXC:

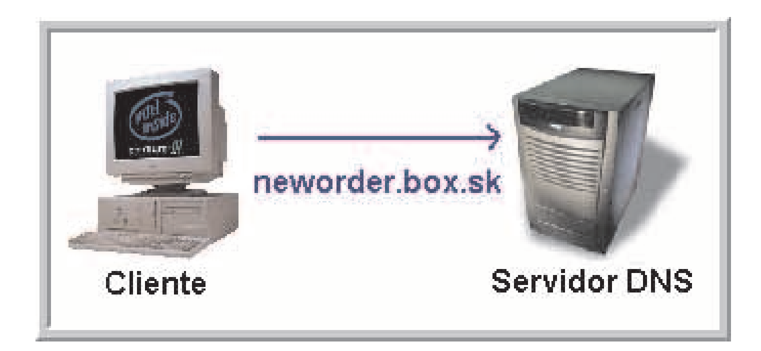
El artículo SERIE RAW: DNS mencionado por el autor ha sido liberado y está disponible en nuestra Web: www.hackxcrack.com

2.1. LAS CAPAS DE UDP/IP

Os recuerdo que el protocolo DNS es el que nos permite utilizar las famosas URLs en lugar de direcciones IP. Es decir, nos permite escribir en nuestro navegador http://www.google.com en lugar de tener que escribir http://216.239.39.99, que es la dirección IP de Google.

Para conseguir esto, existen repartidos por el mundo una serie de servidores encargados de traducir URLs a IPs, e IPs a URLs. Nuestros PCs se comunicarán con alguno de estos servidores cada vez que quieran utilizar una URL, para obtener así su IP correspondiente. Esta comunicación se lleva a cabo a través del protocolo DNS.

Cuando escribimos en nuestro navegador una URL, por ejemplo http://neworder.box.sk, nuestro sistema enviará una consulta DNS a un servidor, indicando en la consulta el nombre que desea traducir (en este caso neworder.box.sk, ya que el prefijo http:// es sólo una indicación al navegador sobre el protocolo a utilizar, pero no forma parte del nombre que deseamos traducir).



El servidor DNS nos enviará una respuesta que contendrá la IP correspondiente a ese nombre.



Gracias a esto, a continuación nuestro navegador podrá acceder a la máquina que contiene la página Web que deseamos visitar, ya que sólo puede existir una comunicación directa entre dos máquinas si cada una conoce la dirección IP de la otra.

Ahora vamos a pensar un poco en cómo se podría conseguir todo este mecanismo del DNS, en el cual un cliente solicita un nombre a un servidor, y el servidor le responde con la IP correspondiente. ¿Qué problemas se nos presentan a la hora de llevar a cabo este proceso aparentemente tan sencillo? Pensadlo un poco, y después mirad la lista de problemas a salvar que os pongo a continuación, para ver si habéis llegado a las mismas conclusiones:

- En primer lugar, por supuesto, hay que conseguir que ambas máquinas (cliente y servidor) tengan una **conexión física**, ya sea por cables, por radio, o por cualquier otro medio físico que les permita establecer una comunicación bidireccional.

- En segundo lugar, sabiendo que en Internet todas las máquinas están conectadas entre sí mediante una compleja red de cables y conexiones inalámbricas, es lógico pensar que será necesario conocer el camino a recorrer en toda esa maraña de cables para **enlazar ambas máquinas** entre sí.

- En tercer lugar, el cliente necesitará conocer la **dirección IP** del servidor DNS, ya que sólo conociendo la dirección IP de una máquina puedes acceder a ella a través de Internet.

- En cuarto lugar, tiene que existir algún mecanismo que le indique al servidor que la consulta que le estamos haciendo es una consulta DNS, y no de cualquier otro tipo. Por ejemplo, el servidor DNS podría ser al mismo tiempo un servidor Web, y un servidor de correo electrónico. Por tanto, tiene que existir un mecanismo que le permita distinguir qué clientes solicitan **servicios** DNS, cuáles solicitan servicios Web, y cuáles solicitan servicios de correo eletrónico.

A grandes rasgos, son cuatro los problemas que hemos encontrado para conseguir llevar a cabo esta comunicación. Y, por supuesto, no es coincidencia que sean cuatro las capas de protocolos utilizadas en una comunicación UDP: **capa física, capa de enlace, capa de red, y capa de transporte**.

Si no fuese gracias a la existencia de estas 4 capas diferenciadas, el protocolo DNS debería encargarse por sí sólo de solucionar todos estos problemas. Es decir, el protocolo DNS debería tener sus propias conexiones físicas entre máquinas, sus mecanismos para encontrar un camino entre todas las máquinas que están conectadas simultáneamente, sus propias direcciones IP, y sus propios mecanismos para diferenciarse de otros servicios (como la Web o el correo electrónico).

Esto convertiría al aparentemente sencillo protocolo DNS en un sistema de una complejidad inabarcable, y lo mismo ocurriría con cualquier otro protocolo que tuviese que

lidiar él solito con todos los problemas existentes en una comunicación.

Vamos a ver entonces cómo se reparte el trabajo de la comunicación para permitir que el protocolo DNS se abstraiga de todo lo que no sea su misión directa.

Empezamos escribiendo una url en nuestro navegador:
http://neworder.box.sk/home.html

En primer lugar, nuestro navegador quitará la parte de la URL que no corresponda al nombre, que es: **neworder.box.sk**.

Teniendo ya el nombre, nuestro sistema construye un paquete que contiene la consulta DNS.

Consulta DNS: neworder.box.sk

Y ahí termina la responsabilidad del protocolo DNS, ya que su única función consiste en enviar consultas de nombres para recibir direcciones IP en respuesta. Por tanto, nuestro sistema pasará ahora la bola a otro protocolo, que será el encargado del **transporte** del paquete DNS.

Existen varios protocolos de transporte, aunque los más importantes son **TCP** y **UDP**.

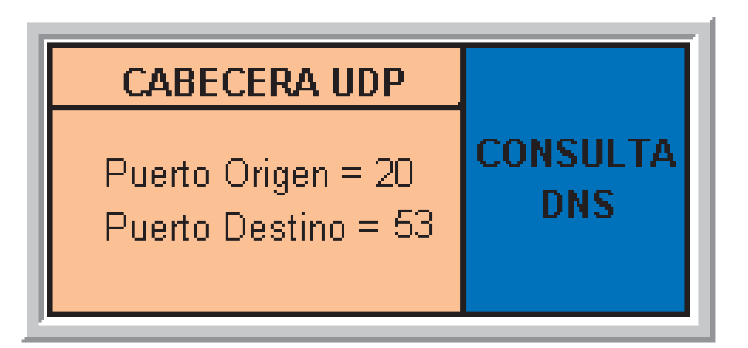
En este caso, el protocolo de transporte utilizado será **UDP**, así que el sistema pasará el paquete DNS al protocolo UDP para que éste realice su función, que consiste básicamente en marcar el paquete con un puerto de origen y otro de destino.

Según el puerto de destino asignado, la máquina que reciba el paquete (el servidor DNS) podrá saber a qué servicio está destinado.

En este caso, el **puerto de destino** es el **53** y, por tanto, es un paquete **DNS**. El **puerto de origen**, en cambio, servirá para que el servidor pueda responder a nuestra consulta, utilizando como puerto de destino el que para nosotros era un puerto de origen.

En resumen, lo que el protocolo UDP ha conseguido es identificar una comunicación entre dos máquinas, entre las cuales podría haber simultáneamente varias comunicaciones establecidas.

Lo que hace UDP es envolver el paquete con una pequeña cabecera que añade la información necesaria, es decir, los puertos.



El trabajo de UDP ya ha terminado, y tiene que pasar el paquete a otro protocolo para que se encargue del resto de problemas de comunicación. Concretamente, el próximo protocolo tendrá que solucionar el problema de identificar a dos máquinas (cliente y servidor) dentro de los millones que hay en toda la **red** Internet. En este caso, el protocolo de red utilizado será el protocolo **IP**.

Este protocolo se encargará de asignar las **direcciones IP** al paquete: la de origen será la nuestra, y la de destino será la del servidor DNS. Igual que sólo podemos llamar a una persona por teléfono si marcamos su número, sólo podremos acceder a una máquina de Internet si "marcamos" su dirección IP.

El protocolo IP, por tanto, añade al paquete una nueva cabecera, que contendrá las IPs de origen y de destino, así como una indicación de que el protocolo desde el cual le llegó el paquete fue el protocolo UDP.

CABECERA IP IP origen = 192.168.1.2 IP destino = 215.22.1.13 Protocolo = UDP	CABECERA UDP	CONSULTA DNS

El paquete pasa ahora al protocolo **Ethernet**, un protocolo que permite encontrar el camino físico para **enlazar** dos máquinas (cada una con una dirección IP que ya conocemos gracias al protocolo IP).

Si cada máquina de Internet tuviese millones de cables para enlazarse con todas y cada una de las demás máquinas conectadas a la red, no sería necesario el uso de protocolos de enlace. Pero, al estar en Internet todo conectado con todo, necesitamos un mecanismo para encontrar el camino entre dos máquinas, aún conociendo las direcciones que las identifican (direcciones IP).

Exactamente lo mismo ocurre en el teléfono, lo cual podemos ver claramente si recordamos a las telefonistas de antaño. Antiguamente (y hoy también ocurre, pero con la diferencia de que está automatizado) en el instante en que marcabas un número de teléfono no se establecía una comunicación. Para conseguir esto hacía falta un procedimiento intermedio, que era el de las telefonistas. Las telefonistas se encargaban de ir conectando y desconectando cables en sus paneles para conseguir enlazar los dos puntos de la comunicación.

En el caso de Internet, a pesar de lo que os hayan contado vuestros padres, no existe una raza de enanitos encargada de conectar y desconectar cables, así que todo esto ocurre de forma automática y virtual, es decir, no se conecta ni se desconecta nada físicamente, si no que simplemente se establecen conexiones lógicas.

Para establecer estas conexiones se utilizan otro tipo de direcciones sobre las que ya hablé un poco en el artículo anterior, y hablaré mucho más en próximos artículos, por lo que de momento nos quedamos sólo con la idea de que el protocolo Ethernet añade su propia cabecera a todas las que había ya en el paquete.

CABECERA ETHERNET MAC origen = 00045A44B899 MAC destino = 009027F69AD4 Protocolo = IP	CABECERA IP	CABECERA UDP	CONSULTA DNS

Hasta ahora ya hemos solucionado los tres últimos problemas que mencioné (volved atrás y revisad la lista de problemas, y veréis que en efecto están solucionados), así que sólo nos queda el primer problema: la **conexión física**. La solución a este problema es bien sencilla: ¡enchufa el cable del módem, porque si no, por mucho protocolo que tengas seguro que no vas a poder enviar ninguna consulta DNS! :-P

Una vez que nuestro paquete está ya circulando por los cables físicos (u ondas de radio físicas), terminará llegando hasta el servidor DNS. El servidor analizará el paquete, verá que se trata de una consulta DNS, y hará lo que tenga que hacer para conseguir el resultado pedido (explicado en detalle en el artículo sobre DNS de la serie RAW, en el número 14 de la revista).

Una vez encontrada la respuesta, tendrá que construir un "sencillo" paquete de respuesta DNS que simplemente contendrá la dirección IP pedida como resultado de la traducción del nombre neworder.box.sk. Este paquete, para poder circular hasta nosotros de vuelta, también tendrá que tener sus propias cabeceras UDP, IP, y Ethernet.

El paquete completo (con las cabeceras) de respuesta podría ser parecido a este:

RESPUESTA DNS:
IP = 66.250.131.132

CABECERA UDP:
Puerto origen = 53
Puerto destino = 20

CABECERA IP:
IP origen = 215.22.1.13
IP destino = 217.12.1.15
Protocolo = UDP

CABECERA Ethernet:
MAC origen = 548F44A5558D
MAC destino = 54F566A47F88
Protocolo = IP

Os recomiendo que volváis atrás para ir comparando cada campo de la respuesta con los campos correspondientes en la consulta. Posiblemente veréis varias cosas que os parecerán muy extrañas. Por ejemplo, ¿por qué la IP de destino de la respuesta no es 192.168.1.2?Esto ya lo explicaré cuando hable sobre el protocolo IP a lo largo del curso. Y, ¿por qué ninguna de las dos direcciones MAC tiene nada que ver con las que había en la consulta? ¡Un poco de paciencia, por favor! ¡Que ya explicaré todo esto a lo largo del curso! :-)

2.2. FUNCIONES DE LA CAPA DE TRANSPORTE UDP

2.2.1. Identificación de conexiones

Lo importante que tenemos que comprender ahora, una vez hecho este repaso al sistema de protocolos por capas UDP/IP, es que gracias al protocolo UDP se puede identificar una comunicación específica entre dos máquinas, gracias a los puertos de origen y de destino.

Esto nos permite tener varias conexiones simultáneas entre dos máquinas (aunque las direcciones IP de cada conexión sean las mismas, las conexiones se pueden diferenciar gracias a los puertos), y también nos permite acceder al mismo servicio en varias máquinas diferentes (podemos, por ejemplo, estar viendo dos páginas Web de dos servidores diferentes al mismo tiempo).

Supongo que la necesidad de especificar un puerto de destino en cada paquete os habrá quedado muy clara, pero quizá no tenéis tan clara la necesidad de especificar también un puerto de origen.

Imaginemos que nuestro navegador realiza una consulta DNS con nuestro servidor DNS, pero casi al mismo tiempo nuestro cliente de correo electrónico también está realizando otra consulta DNS al mismo servidor. No podemos saber de ninguna manera cuál de las dos respuestas nos llegará antes.

Por una parte, si leísteis el artículo sobre DNS de la serie RAW, sabréis que el tiempo para procesar una consulta DNS puede variar enormemente. Por ejemplo, una traducción que se encuentre en la cache del servidor DNS nos llegará casi al instante, mientras que otra traducción menos afortunada podría requerir establecer varias conexiones con varias máquinas, desde los servidores raíz, y los servidores del TLD, hasta el último servidor de la jerarquía DNS. Si no sabéis de qué hablo no os asustéis, que esto no tiene nada que ver con UDP, si no con DNS, y no os hace falta conocerlo ahora. :-)

La idea con la que os tenéis que quedar es que de ninguna manera se puede asumir que al hacer dos consultas una detrás de otra, las dos respuestas nos llegarán en el mismo orden. Por tanto, si el servidor nos envía las dos respuestas, necesitamos de alguna manera saber cuál de las dos es la que solicitó nuestro navegador, y cuál es la que solicitó nuestro cliente de correo electrónico. ¡De ahí la necesidad del puerto de origen!

Por supuesto, si leísteis el artículo sobre DNS os estaréis preguntando: ¿y qué hay de los transaction ID? ¡Permiten perfectamente diferenciar una consulta DNS de cualquier otra! Pues sí, es cierto, pero el caso de DNS es un poco especial, ya que no es normal que los protocolos que utilicen UDP (o TCP) para el transporte incluyan sus propios identificadores para cada conexión. Si se hizo así en el caso de DNS no fue por motivos de identificar cada conexión, ya que para eso

sobraba con lo que hace UDP, si no por motivos de seguridad, como ya vimos al hablar en la serie RAW sobre los ataques por envenenamiento de DNS.

En la práctica, hay casos en los que el puerto de origen es irrelevante, por lo que no siempre es obligatorio especificarlo. En estos casos, lo que se hace es usar el puerto 0 como origen.

2.2.2. Corrección en los datos

Imaginad que por cualquier problema en la "línea" los datos llegasen corruptos hasta nosotros, y uno de los números que forman la IP que hemos solicitado al servidor DNS es incorrecto.

Sería un gran problema no tener una cierta seguridad de que los datos que nos llegan de un servidor DNS son correctos. Estaríamos cada dos por tres estableciendo conexiones con direcciones IP incorrectas.

UDP no puede garantizar que los datos lleguen, pero si que nos da cierta garantía de que, si han llegado, son correctos. Para ello incorpora en su cabecera un dato que no hemos visto antes (ya que estabamos viéndolo sólo a grandes rasgos), que nos permite comprobar que los datos son correctos, tal y como veremos más adelante.

Este problema de la corrección de los datos perfectamente podría haberlo incluido en la lista de problemas de la comunicación, pero no lo hice a propósito para que saliesen justo cuatro. :-P

2.3. LO QUE NO NOS DA EL PROTOCOLO UDP FRENTE A TCP

2.3.1. Fiabilidad en la comunicación

Aunque antes desmentí el mito de los enanitos telefonistas de Internet, unos enanos que sin duda si que existen en la red son los enanos gruñones que se dedican a desconectar cables de vez en cuando, ocasionando así la pérdida de paquetes.

No cabe duda de que ninguna tecnología es perfecta, y por eso siempre puede haber pérdidas de datos en cualquier comunicación. ¿Qué ocurriría, por ejemplo, si no nos llegase la respuesta del servidor DNS, a pesar de que éste la hubiera enviado? Si, por cualquier motivo, la respuesta se perdiese por el camino, no pudiendo llegar hasta nosotros, ¿habría alguna forma de detectar y solventar esta situación?

Analizando el funcionamiento básico del protocolo UDP, tal y como lo hemos visto, no habría ninguna manera. En UDP cada máquina envía sus paquetes a la red, sin tener ninguna certeza de si llegarán o no a su destino. Una vez que el servidor DNS envíe su respuesta, se desentenderá del asunto, al no tener forma de saber si la respuesta nos ha llegado o no.

Este es el principal problema de los protocolos **no orientados a conexión**, como es el caso de UDP. Como ya vimos en el artículo sobre DNS, en el caso de TCP esto es muy diferente, ya que TCP es un protocolo **orientado a conexión**. Por cada paquete transmitido en TCP es obligatorio que el receptor envíe un acuse de recibo para que el emisor sepa con certeza que los datos han llegado al destino. Esto **no** es así en UDP.

En UDP no hay manera de saber si los datos llegan al destino. Esto es una gran desventaja pero, por otra parte, tiene la gran ventaja de hacer la comunicación mucho más fluida, al no verse interrumpida constantemente por miles de paquetes de acuse de recibo.

Esta característica convierte a UDP en un protocolo de transporte ideal para comunicaciones sencillas (como DNS, o TFTP (Trivial File Transfer Protocol --- Protocolo Trivial de Transferencia de Ficheros)), y para envíos masivos de flujos de datos (como en

transmisiones multimedia de vídeo, audio, ...), pero en un protocolo que de ninguna manera sirve para comunicaciones que requieren fiabilidad (FTP, IRC, HTTP, ...).

Por ejemplo, ahora que menciono el IRC, imaginemos lo que sería una conversación en un chat en el que no tuviésemos la certeza de que lo que decimos vaya a llegar al resto de interlocutores. Algunas frases se perderían por el camino, y nadie podría saberlo, por lo que muchas conversaciones perderían su sentido y, probablemente, se convertirían en diálogos de besugos (aunque, incluso utilizando TCP hay gran cantidad de diálogos de besugos en IRC, pero eso más que un problema tecnológico es un problema intelectual).

2.3.2. Flujo ordenado de datos

Otro gran problema potencial de UDP es el del **desorden**. En el caso de DNS parece todo muy sencillo: una consulta -> una respuesta. Pero, ¿qué ocurriría si, por ejemplo, intentásemos hacer un chat mediante UDP?

Internet es una red increíblemente compleja, y existen muchos factores que pueden determinar que los paquetes vayan más o menos rápidos en cada instante. Esto puede dar lugar a situaciones tan extrañas como que dos paquetes salgan uno tras otro de una máquina, pero primero llegue al receptor el último que salió. Si no hay un mecanismo para controlar el orden en que deben ir los paquetes, en el chat veríamos respuestas a preguntas aún no formuladas, monólogos sin sentido, etc.

Otros protocolos de transporte, como TCP, sí que incluyen mecanismos para garantizar el correcto orden de los paquetes; pero no es el caso de UDP, por lo que es otro aspecto a tener en cuenta a la hora de decidir si nuestra aplicación funcionará mejor sobre un protocolo seguro y robusto como es TCP, o sobre un protocolo más fluido como es UDP.

2.3.3.Tratamiento adecuado de los paquetes grandes

Imaginemos que enviamos una consulta de DNS a un servidor, y éste nos responde, pero su respuesta se ha perdido por el camino. En una situación normal, tras pasar un tiempo sin recibir la respuesta, volveríamos a solicitarla, asumiendo que se perdió por el camino. ¡Vaya! Eso sí que es genial. Acabamos de solucionar el problema de la fiabilidad en la comunicación. ¿O no?...

¿Que ocurriría si en lugar de ser una simple consulta DNS nuestro paquete fuese una solicitud a un servidor FTP para bajarnos un archivo grande (por ejemplo una ISO de 700MB?

Si se perdiese el paquete de respuesta, el servidor tendría que volver a enviar nada menos que 700MB! Esto, por supuesto, es una barbaridad, y lo ideal para minimizar este tipo de problemas es dividir los paquetes demasiado grandes en paquetes más pequeños, para que luego estos se unan en el orden adecuado al llegar al destino, y reconstruir así el paquete original. Si ha habido algún fallo tecnológico en un momento dado, normalmente sólo se perderá alguno de los paquetes pequeños, que podrá ser reenviado sin problemas.

UDP no incorpora ningún mecanismo para partir los paquetes grandes, por lo que tampoco es adecuado para esta clase de comunicaciones. Quizá ahora os estaréis preguntando por qué entonces he dicho que UDP puede ser adecuado para transmisiones multimedia donde, sin duda, el flujo de datos es enorme.

La cuestión fundamental aquí es que en los contenidos multimedia no es crítica la pérdida de los datos. Por ejemplo, podemos estar transmitiendo una película a razón de 20 fotogramas por segundo, enviando por ejemplo cada fotograma en un paquete independiente. En este caso, la pérdida de un único paquete

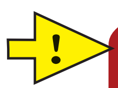
no sería en absoluto crítica. Simplemente perderíamos un fotograma, y eso sería prácticamente imperceptible para el ojo humano, y para nada afectaría al espectador de la película. En cualquier caso, nunca sería necesario reenviar el paquete.

El tema de la partición de paquetes grandes es mucho más complejo de lo que puede parecer en un principio ya que, entre otras cosas, no sólo es misión del protocolo de nivel de transporte, pero mejor que me quede calladito ahora, que si no os voy a liar bastante, y ya habrá tiempo de ver todo eso a lo largo del curso. ;-)

3. UDP EN DETALLE

Antes de deciros cuál es el RFC que detalla el protocolo UDP os tengo que poner sobre aviso: podéis asustaros del tamaño del RFC, así que no tengáis miedo, que yo haré todo lo posible por resumiros lo más importante, como siempre. ;-)

El RFC en cuestión es el **RFC 768** (ftp://ftp.rfc-editor.org/in-notes/rfc768.txt). Ejem... esto.... ¿sólo dos páginas? Bueno, quizá he exagerado un pelín con eso del tamaño. 0;-)



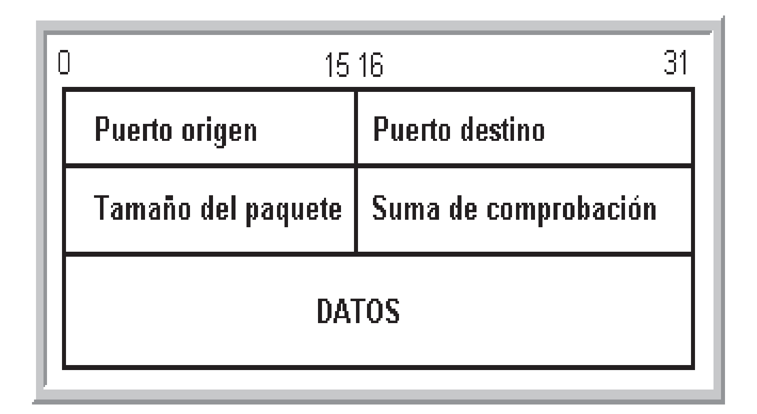
NOTA DE PC PASO A PASO / HXC

Para los que nunca nos han leido, decir que RFC (**R**equest **F**or **C**omments) son una serie de más de 3000 documentos donde se detalla todo lo relacionado con la tecnología de Internet.

Para los ALERGICOS al inglés, tienes al final de este artículo el RFC en perfecto castellano ;)

El RFC de UDP está sin duda escrito para gente entendida en el tema, por lo que no se andan con explicaciones, van directamente al grano, que es la especificación de las cabeceras UDP.

Como ya hemos explicado los conceptos, podemos permitirnos el lujo de poner las cabeceras sin explicar nada, como hace el RFC :)



Esta es la estructura de una cabecera UDP, es decir, del trozo que se añade a cada paquete para luego pasarlo al protocolo IP, el cual a su vez añadirá otro trozo, que se sumará también al trozo que luego añadirá el protocolo Ethernet.

En los ejemplos anteriores veíamos sólo dos campos: puerto de origen, y puerto de destino. Aquí no sólo vemos todos los campos, si no que además vemos su longitud.

Los "numerajos" que he puesto en la ilustración encima de la cabecera son el número de bits. Es decir, el campo **Puerto origen** abarca desde el bit 0 hasta el bit 15, es decir, 16 bits, que son 2 bytes. Por tanto, como 2^{16} son **65536**, éste es el número de puertos que existen en UDP (y también en TCP, por cierto).

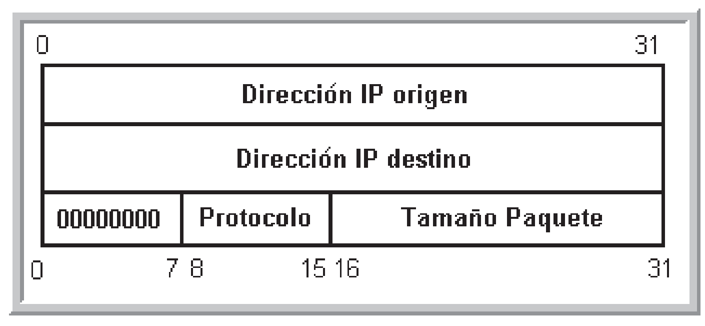
Como vemos, el campo **Puerto destino** abarca desde el bit 16 hasta el 31, que son otros 16 bits.

El campo **Tamaño paquete** son otros 16 bits, así como el campo **Suma de comprobación**.

En el campo **DATOS** es donde se encuentra todo lo que ha englobado el paquete UDP. Por ejemplo, en el caso de DNS, la consulta DNS estaría en el campo **DATOS** del paquete UDP. Su tamaño ahora puede ser cualquiera, y precisamente para eso sirve el campo **Tamaño paquete**.

Lo he llamado **Tamaño paquete** porque es un nombre más claro, pero su nombre original es **Length** (por si leéis algo sobre el tema y os líais). Este campo indica el número de bytes que ocupa todo el paquete UDP, incluyendo la cabecera. Al ser la cabecera de un tamaño fijo de 64 bits, es decir, 8 bytes, el campo **Length** será siempre como mínimo 8 (sería el caso de un paquete que no contenga datos).

Con respecto al campo **Suma de comprobación** (**checksum** en el original) hay bastante más que decir. Éste es el campo que nos permite comprobar que los datos que hemos recibido son correctos, es decir, que el paquete no ha tenido problemas que hayan corrompido los datos durante su transporte. Para calcular la Suma de comprobación se genera una cabecera con los siguientes campos:



Todos estos campos los conocemos, excepto el campo **Protocolo**. Cada protocolo existente tiene un número único asignado que le diferencia de los demás protocolos, y esto permite por ejemplo, propagar la información sobre a qué protocolo hay que pasar un paquete después de procesarlo, tal y como hemos visto en el ejemplo al principio de este artículo, en el cual la cabecera IP contenía un campo **Protocolo = UDP**, y la cabecera Ethernet contenía un campo **Protocolo = IP**.

Esta lista está mantenida por el **IANA** (**I**nternet **A**ssigned **N**umbers **A**uthority), así que podéis consultarla en www.iana.org. Si queréis una lista más sencilla, aunque no actualizada, la tenéis en el **RFC 1700** (http://www.rfc-editor.org/rfc/rfc1700.txt).

Si habéis echado un vistazo a la lista, habréis visto que el protocolo **UDP** tiene asignado el número **17**. Por tanto, en la cabecera que estamos tratando ahora habría que poner un 17 en el campo Protocolo.

Con todos estos campos se realiza una operación de aritmética binaria, que es la suma en complemento a uno de 16 bits de toda la cabecera. No voy a entrar en detalles de cómo se realiza esta operación, así que si tenéis curiosidad podéis consultar **el RFC 1071** o, si lo preferís, tenéis aquí directamente un código en C que calcula la suma de comprobación (checksum) para el protocolo TCP:
http://www.netfor2.com/tcpsum.htm.

Si queréis utilizar este código para el caso de UDP sólo tenéis que cambiar esta línea:

```
u16 prot_tcp=6;
```

Por esta otra:

```
u16 prot_tcp=17;
```

Aunque, si queréis que quede un poco más bonito, preocupaos de cambiar también el nombre de la variable para que se llame **prot_udp**. ;-)

Cuando recibimos un paquete UDP, nuestro sistema compara la suma de comprobación con los datos que conoce sobre el paquete (IP de origen, IP de destino, y tamaño de paquete UDP), y si algo no concuerda sabrá que ha habido algún problema y los datos del paquete no son válidos.

Un detalle curioso con respecto al complemento a uno (la forma de representación binaria utilizada en la suma de comprobación) es que existen dos formas de representar el cero: o bien que todos los dígitos sean 1, o bien que todos sean 0.

Esto nos es útil en nuestro caso, ya que lo que se hace es usar la primera representación del cero (todos los dígitos son 1) para las

sumas cuyo resultado son cero, y la otra representación del cero (todos los dígitos son 0) indica simplemente que ese paquete no incluye una suma de comprobación, por el motivo que sea (por ejemplo, si el protocolo no requiere una comprobación de los datos).

4. UDP EN LA PRÁCTICA

Aquí voy a presentar algunas herramientas que nos pueden ser útiles a lo largo de todo el curso de TCP/IP. Intentaré no discriminar a nadie, explicando herramientas tanto para Windows como para Linux.

Concretamente, las herramientas que vamos a utilizar nos permiten construir paquetes desde cero para varios protocolos diferentes (en este caso, el que nos interesa es UDP).

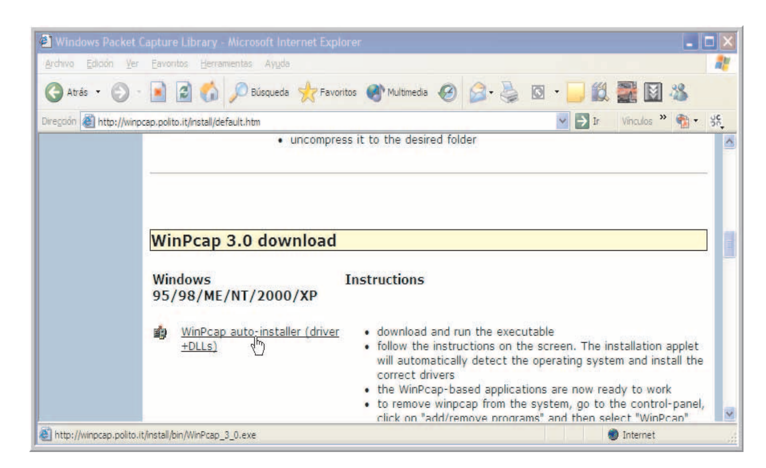
Para que estos programas funcionen, nuestro sistema tiene que permitir el uso de **raw sockets**, es decir, de paquetes generados desde cero. Linux siempre ha permitido los raw sockets, y Windows sólo lo hace desde sus versiones XP y 2000. Si aún queda algún usuario de Windows 9x (95, 98, o Millenium), que tampoco se preocupe, ya que puede utilizar las librerías **WinPCap** para que su sistema admita raw sockets. Estas librerías las tenéis en: http://winpcap.polito.it/.

De todas maneras, nosotros vamos a utilizar para nuestras prácticas en Windows un programita llamado **Nemesis.** Esta aplicación necesitará las librerías WinPCap incluso si tienes un Windows XP y 2000... tranquilos, ahora veremos detalladamente todo esto :)

4.1. NEMESIS PARA WINDOWS (TODAS LAS VERSIONES)

En primer lugar vamos con los usuarios de Windows, no porque sean más importantes, si no porque ya he empezado hablando de ellos antes. :-P

Lo primero que tenemos que hacer es bajarnos las librerías **WinPCap**. Para ello abriremos nuestro navegador preferido e iremos a http://winpcap.polito.it/, pincharemos en DOWNLOADS (en el menú de la izquierda) y aparecerá una lista descargas. Tenemos que descargarnos el WINPCAP 3.0

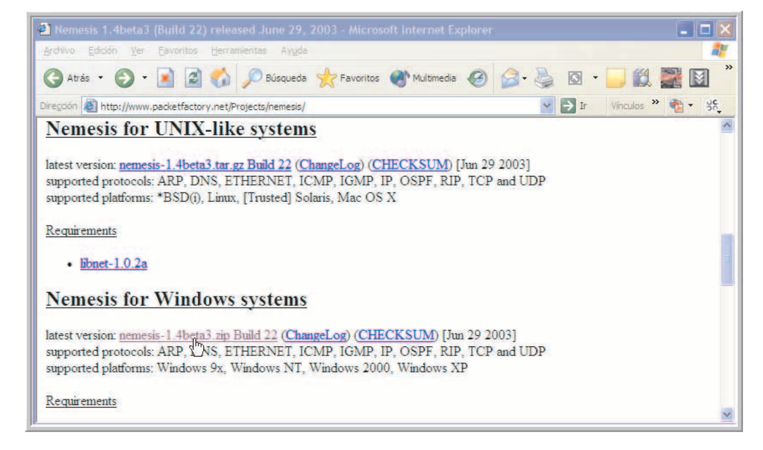


No cometas el error de bajarte la última versión (WINPCAP 3.1 BETA), hemos podido comprobar que no funciona correctamente en bastantes de nuestros equipos, incluso con configuraciones muy "normalitas". Así que ya sabes, descarga la versión 3.0 tal y como te indicamos :)

Una vez descargado el archivo mencionado (WinPcap_3_0.exe) lo ejecutaremos para que se instale en el sistema. Si te pide reiniciar Windows, hazle caso :)

A continuación, ya podemos bajar el **Nemesis**, que lo tenéis aquí:

http://www.packetfactory.net/Projects/nemesis/ (Por supuesto, bajad la versión para Windows).



Obtendremos el archivo comprimido nemesis-1.4beta3.zip y lo descomprimiremos en la carpeta c:\nemesis. Este programa no necesita instalarse, ahora veremos cómo utilizarlo ;)

Podemos pasar ya a la acción. Si vais al directorio (carpeta) sobre donde habéis descomprimido el Nemesis (en nuestro caso c:\nemesis), veréis que hay una serie de archivos **TXT**, uno de los cuales se llama **nemesis-udp.txt**. En ese archivo tenéis instrucciones para utilizar Nemesis con UDP, que es lo que nos interesa ahora. Enseguida descubriréis que Nemesis es una aplicación más práctica que estética, ya que no tiene interfaz gráfica, si no que funciona desde la consola **MS-DOS**.

Vamos a ver un primer ejemplo de uso de Nemesis:

Primero abrimos una consola de MS-DOS (una de nuestras habituales "ventanitas negras"). ¿Cómo se hace eso? Ya lo hemos explicado mil veces, tienes dos opciones:

- Menu Inicio --> Todos los Programas --> Accesorios --> Símbolo del sistema

- Menu Inicio --> Ejecutar y en la ventana que nos aparezca escribimos command y pulsamos enter

Ahora, utilizando nuestra consola de MS-DOS vamos al directorio donde hemos tenemos en Nemesis esperándonos. Para el que no sepa hacer eso (que a estas alturas de la publicación ya deberían ser MUY POCOS):

- En nuestra ventanita negra escribimos **cd c:** y pulsamos enter (con esto nos vamos al directorio raíz del disco duro C).

- En nuestra ventanita negra escribiremos **cd c:\nemesis** y pulsaremos enter (con esto nos posicionamos en el directorio donde tenemos nuestro Nemesis).

- Escribimos **dir** y pulsamos enter. Con ello veremos un listado del contenido de la carpeta Nemesis, veremos, entre otros el archivo nemesis.exe (nuestro preciado Nemesis ;))

```
Símbolo del sistema
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\RUBEN>cd c:\

C:\>cd nemesis

C:\NEMESIS>dir
 El volumen de la unidad C no tiene etiqueta.
 El número de serie del volumen es: 1871-9F3A

 Directorio de C:\NEMESIS

14/03/2004  00:53    <DIR>          .
14/03/2004  00:53    <DIR>          ..
29/06/2003  20:41               701 INSTALL.txt
28/05/2003  21:54            64.762 LibnetNT.dll
29/06/2003  20:37             3.718 nemesis-arp.txt
29/06/2003  20:37             8.870 nemesis-dns.txt
29/06/2003  20:37             3.134 nemesis-ethernet.txt
29/06/2003  20:37             9.186 nemesis-icmp.txt
29/06/2003  20:37             5.822 nemesis-igmp.txt
29/06/2003  20:37            10.723 nemesis-ip.txt
29/06/2003  20:37             6.105 nemesis-ospf.txt
29/06/2003  20:37             9.143 nemesis-rip.txt
29/06/2003  20:37             6.824 nemesis-tcp.txt
29/06/2003  20:37             5.374 nemesis-udp.txt
29/06/2003  20:29           136.533 nemesis.exe
29/06/2003  20:37             1.667 nemesis.txt
29/06/2003  20:38             2.688 README.txt
              15 archivos        275.250 bytes
               2 dirs     665.112.576 bytes libres

C:\NEMESIS>
```

Por fin, escribimos:

nemesis udp –v –y 53 –D 194.224.52.6 –P –

y pulsamos enter, nos aparecerá algo como esto:

UDP Packet Injection -=- The NEMESIS Project Version 1.4beta3 (Build 22)

[IP] 68.253.120.0 > 194.224.52.6
[IP ID] 64988
[IP Proto] UDP (17)
[IP TTL] 255
[IP TOS] 00
[IP Frag offset] 0000
[IP Frag flags]

[UDP Ports] 64988 > 53

A continuación, escribimos cualquier cosa:

hola
Wrote 33 byte UDP packet.

UDP Packet Injected

Vamos a analizar lo que hemos hecho.

En primer lugar, al llamar a Nemesis con la opción **udp** en primer lugar (**nemesis udp**) le decimos que queremos enviar un paquete **UDP**, de entre todos los protocolos que soporta Nemesis.

La opción **–v** es la clásica opción **verbose** de muchos programas, que nos permite ver en pantalla información detallada sobre lo que estamos haciendo.

La opción **–y** sirve para detallar el **puerto UDP de destino**. En este caso, hemos utilizado el conocido puerto 53, que es el puerto de DNS.

La opción **–D** sirve para especificar la **dirección IP de destino**. En este caso, hemos utilizado la dirección de un servidor DNS de Telefónica.

Por último, la opción **–P** sirve para indicar qué datos queremos meter en el campo **DATOS** del paquete UDP. Por ejemplo, en este caso, debería ir aquí la supuesta consulta DNS. En el caso de este ejemplo, poniendo **–P –** indicamos a la aplicación que queremos meter nosotros los datos directamente desde teclado.

Analicemos ahora el comportamiento de Nemesis. En primer lugar, vemos que ha utilizado como IP de origen una que no es la nuestra, por lo que la respuesta a este paquete difícilmente podría llegarnos a nosotros. Es más, es también improbable que el paquete llegue siquiera a su destino, ya que los routers que habrá en los primeros pasos del camino (por ejemplo, nuestro router ADSL, o el router de nuestro ISP) probablemente rechazarán el paquete al no provenir supuestamente de una de las máquinas a las que tienen que dar servicio.

¿Y cual puede ser la utilidad de utilizar una IP que no es la tuya? Pues ya hablaremos sobre todo eso a lo largo del curso, pero os adelanto que esto puede servir para llevar a cabo gran número de ataques.

En nuestro caso no queremos atacar a nadie, simplemente queremos comprender de forma práctica el funcionamiento del protocolo UDP. Por lo tanto preferimos utilizar nuestra IP como IP de origen:

nemesis udp –v –y 53 –S 192.168.1.2 –D 194.224.52.6 –P –

Hemos añadido la opción **–S** para especificar la **IP de origen**. Si estamos detrás de un router, tendremos que especificar nuestra IP privada (de nuestra red local), ya que el router ya se encargará de convertirla luego en vuestra IP pública.



Ya se explico...

Ya se ha explicado mil veces, pero bueno… … para saber cuál es tu IP, simplemente abre una "ventanita negra", escribe **ipconfig /all** y pulsa enter. Este tema ya se trató en profundidad en los primeros números de Hack x Crack.

Si seguimos observando el comportamiento de Nemesis, veremos que ha escogido un puerto aleatorio como **puerto de origen**. Para la mayoría de las aplicaciones esto nos valdrá, pero si queremos probar a especificar un puerto de origen en concreto utilizaremos la opción **–x**:

nemesis udp –v –x 1000 –y 53 –S 192.168.1.2 –D 194.224.52.6 –P –

Esto puede sernos útil si estamos detrás de un firewall muy exigente que sólo nos permita utilizar algunos puertos UDP determinados (en el caso del ejemplo, el puerto 1000).

Por último, os habréis dado cuenta de que no es nada práctico meter los datos directamente desde teclado, ya que es bastante complicado generar una consulta DNS a pelo e introducirla desde el teclado. En la mayoría de los casos, os será mucho más útil construir primero la consulta con algún editor hexadecimal (por ejemplo, podéis utilizar UltraEdit, que además de ser un magnífico editor y visor de textos, es también un editor hexadecimal básico), guardarla en un **fichero**, y luego cargarla directamente en Nemesis con la opción **–P**. Por ejemplo, si nuestro fichero se llama **consulta.txt** haremos:

nemesis udp –v –x 1000 –y 53 –S 192.168.1.2 –D 194.224.52.6 –P consulta.txt

Podemos, por ejemplo, capturar una consulta DNS real con un sniffer, después modificarla a nuestra voluntad con el editor hexadecimal, guardar la consulta modificada en un archivo, y después enviarla a través de Nemesis.

En el artículo de la Serie RAW sobre DNS hablé acerca de la técnica de **envenenamiento de cache DNS**. Con Nemesis, y un sencillo shell script podríamos explotar esta técnica una vez conseguidos los datos necesarios. Si no habéis leído el artículo sobre DNS, os recomiendo que paséis este punto, porque probablemente no os enteraréis de nada. :-)

Si, por ejemplo, tenemos estos datos para llevar a cabo el ataque:

- **Dirección IP del servidor DNS de la víctima** = 194.224.52.6
- **Dirección IP del servidor DNS authoritative que queremos suplantar** = 194.220.51.2
- **Puerto UDP utilizado por el servidor DNS de la víctima** = 1200

Suponemos, por supuesto, que conocemos también el identificador de transacción, y que con él hemos construido una respuesta DNS falsa que tenemos almacenada en el archivo **envenenamiento.txt**. La forma de lanzar esta respuesta falsa sería:

nemesis udp –x 53 –y 1200 –S 194.220.51.2 –D 194.224.52.6 –P envenenamiento.txt

Podemos automatizar esto mediante un script que haga un bucle en el cual vaya utilizando distintos Transaction ID y, tal y como vimos en el artículo sobre DNS, según la versión de BIND que utilice el servidor DNS de la víctima tendremos una mayor o menor probabilidad de éxito.

4.2. HPING PARA LINUX

Existe también versión de Nemesis para Linux, pero os enseñaré mejor una herramienta bastante más potente, que es **Hping**. Podéis bajar Hping de www.hping.org. Para instalarlo tendréis que compilarlo primero. Los pasos a seguir son los típicos:

- Primero, una vez bajado el archivo con extensión **.tar.gz**, lo descomprimimos y desempaquetamos con el comando: **tar -zxvf hping2.0.0-rc2.tar.gz**
- Nos creará un directorio **hping2-rc2**, en el cual entraremos (**cd hping2-rc2**), para después ejecutar: **./configure**
- A continuación, ejecutamos: **Make**
- Y, por último: **make install**

En caso de cualquier problema con este sistema estándar de instalación, podéis consultar el archivo **INSTALL** con instrucciones más detalladas sobre el proceso.

Una vez instalado, vamos a probar un poco su funcionamiento. Por supuesto, tendréis la correspondiente página del manual de hping:

man hping2

Hping no sólo permite enviar un único paquete,

si no que además implementa sencillos bucles para poder generar muchos paquetes sin necesidad de programar scripts. Si queremos que envíe un único paquete tendremos que usar la opción --count 1. Aunque mejor que veamos directamente un ejemplo:

hping2 193.224.52.6 --udp --destport 53 --file envenenamiento.dat --data 14 --count 1

El primer parámetro (**193.224.52.6**), como podéis imaginar, es la **IP de destino** del paquete, y es el único parámetro obligatorio.

El parámetro **--count** ya hemos dicho que indica el **número de paquetes** a enviar. Según las opciones estos paquetes podrán ser diferentes entre sí, por ejemplo, incrementando el puerto de origen en cada paquete. Si queréis más detalle sobre estas opciones consultad la página del manual. Por defecto, el puerto de origen se incrementa con cada paquete, así que si queremos utilizar **siempre el mismo puerto** utilizaremos la opción **--keep**.

El parámetro **--udp**, por supuesto, indica que el protocolo a utilizar será **UDP**.

El parámetro **--destport** es el **puerto de destino** del paquete.

El parámetro **--file** es el **fichero** en el que tenemos el campo **DATOS** del paquete, es decir, por ejemplo la consulta DNS, o la respuesta falsa para el caso de que estemos haciendo un ataque de envenenamiento de cache DNS.

El parámetro **--data** es el tamaño de los datos sin la cabecera, es decir, en este caso sería igual al campo **Tamaño paquete** de la cabecera UDP, pero **restándole 8 bytes**, que son los que ocupa la cabecera UDP.

Si quisiésemos especificar un **puerto de origen** usaríamos el parámetro **--baseport**. Si no se especifica, el puerto de origen será aleatorio.

Una opción curiosa de hping es la opción **--badcksum** que genera una **suma de comprobación inválida** en el paquete enviado, lo cual puede ser útil para comprobar la reacción de un sistema ante un paquete malformado.

A lo largo del curso, entraremos en más detalle en el funcionamiento de éstas y otras herramientas. Por el momento, os animo a que vayáis investigando por vuestra cuenta.

Autor: PyC (LCo)

```
RFC 768                                          J. Postel
                                                       ISI
                                      28 de Agosto de 1980

        PROTOCOLO DE DATAGRAMAS DE USUARIO
             (User Datagram Protocol)

  (Traducción al castellano: Diciembre de 1999)
 (Por Domingo Sánchez Ruiz <domingo@quark.fis.ucm.es>)

Introducción

      Este Protocolo de Datagramas de Usuario (UDP:
      User Datagram Protocol)se define con la
      intención de hacer disponible un tipo de
      datagramas para la comunicación por intercambio
      de paquetes entre ordenadores en el entorno
      de un conjunto interconectado de redes de
      computadoras.

      Este protocolo asume que el Protocolo de
      Internet (IP: Internet protocol) [1] se
      utiliza como protocolo subyacente.

      Este protocolo aporta un procedimiento para
      que los programas de aplicación puedan enviar
      mensajes a otros programas con un mínimo de
      mecanismo de protocolo.  El protocolo se
      orienta a transacciones, y tanto la entrega
      como la protección ante duplicados no se
      garantizan.
      Las aplicaciones que requieran de una entrega
      fiable y ordenada de secuencias de datos
      deberían utilizar el Protocolo de Control de
      Transmisión (TCP: Transmission Control
      Protocol). [2]
```

```
Formato
      0        7 8      15 16     23 24      31
      +--------+--------+--------+--------+
      |    Puerto de    |    Puerto de    |
      |      Origen     |     Destino     |
      +--------+--------+--------+--------+
      |                 |                 |
      |    Longitud     | Suma de Control |
      +--------+--------+--------+--------+
      |
      |          octetos de datos ...
      +---------------- ...
```

Formato de la Cabecera de un Datagrama de Usuario

Campos

El campo Puerto de Origen es opcional; cuando tiene sentido, indica el puerto del proceso emisor, y puede que se asuma que ése sea el puerto al cual la respuesta debería ser dirigida en ausencia de otra información. Si no se utiliza, se inserta un valor cero.

El campo Puerto de Destino tiene significado dentro del contexto de una dirección de destino en un entorno internet particular.

El campo Longitud representa la longitud en octetos de este datagrama de usuario, incluyendo la cabecera y los datos. (Esto implica que el valor mínimo del campo Longitud es ocho.)

El campo Suma de Control (Checksum) es el complemento a uno de 16 bits de la suma de los complementos a uno de las palabras de la combinación de una pseudo-cabecera construida con información de la cabecera IP, la cabecera UDP y los datos, y rellenada con octetos de valor cero en la parte final (si es necesario) hasta tener un múltiplo de dos octetos.

La pseudo-cabecera que imaginariamente antecede a la cabecera UDP contiene la dirección de origen, la dirección de destino, el protocolo y la longitud UDP. Esta información proporciona protección frente a datagramas mal encaminados. Este procedimiento de comprobación es el mismo que el utilizado en TCP.

```
 0      7 8     15 16    23 24     31
+--------+--------+--------+--------+
|        dirección de origen        |
+--------+--------+--------+--------+
|        dirección de destino       |
+--------+--------+--------+--------+
|  cero  |protocol|  longitud UDP   |
+--------+--------+--------+--------+
```

Si la suma de control calculada es cero, se transmite como un campo de unos (el equivalente en la aritmética del complemento a uno). Un valor de la suma de control trasmitido como un campo de ceros significa que el emisor no generó la suma de control (para depuración o para protocolos de más alto nivel a los que este campo les sea indiferente).

Interfaz de Usuario

Un interfaz de usuario debería permitir:

la creación de nuevos puertos de recepción, operaciones de recepción en los puertos de recepción que devuelvan los octetos de datos y una indicación del puerto de origen y de la dirección de origen, y una operación que permita enviar un datagrama, especificando los datos y los puertos de origen y de destino y las direcciones a las que se debe enviar.

Interfaz IP

El módulo UDP debe ser capaz de determinar las direcciones de origen y destino en un entorno internet así como el campo de protocolo de la cabecera del protocolo internet. Una posible interfaz UDP/IP devolvería el datagrama de internet completo, incluyendo toda la cabecera, en respuesta a una operación de recepción. Un interfaz de este tipo permitiría también al módulo UDP pasar un datagrama deinternet completo con cabecera al módulo IP para ser enviado. IP verificaría ciertos campos por consistencia y calcularía la suma de control de la cabecera del protocolo internet.

Aplicación del Protocolo

Los usos principales de este protocolo son el Servidor de Nombres de Internet [3] y la Transferencia Trivial de Ficheros (Trivial File Transfer) [4].

Número del protocolo

Este es el protocolo 17 (21 en octal) cuando se utilice en el Protocolo de Internet (IP). Se indican otros números de protocolo en [5].

Referencias

[1] Postel, J., "Internet Protocol," RFC 760, USC/Information Sciences Institute, Enero de 1980. (Nota del T. Hay traducción al español por P.J. Ponce de León: "Protocolo Internet", Mayo 1999.)

[2] Postel, J., "Transmission Control Protocol," RFC 761, USC/Information Sciences Institute, Enero de 1980.

[3] Postel, J., "Internet Name Server," USC/Information Sciences Institute, IEN 116, Agosto de 1979.

[4] Sollins, K., "The TFTP Protocol," Massachusetts Institute of Technology, IEN 133, Enero de 1980.

[5] Postel, J., "Assigned Numbers," USC/Information Sciences Institute, RFC 762, Enero de 1980.

Nota del traductor

Este documento y las traducciones al español mencionadas en las referencias pueden encontrarse en: http://lucas.hispalinux.es/htmls/estandares.html

El proyecto de traducción de RFC al español tiene su web de desarrollo en: http://www.arrakis.es/~pjleon/rfc-es

EL GANADOR DEL SORTEO DE UN SUSE LINUX 9 DEL MES DE FEBRERO ES:

MANUEL PEREZ GARCIA

SEVILLA

SUSCRIBETE A PC PASO A PASO

SUSCRIPCIÓN POR:
1 AÑO
11 NUMEROS

=

45 EUROS (10% DE DESCUENTO)
+
SORTEO DE UNA CONSOLA XBOX
+
SORTEO 2 JUEGOS PC (A ELEGIR)

Contra Reembolso

Solo tienes que enviarnos un mail a preferente@hackxcrack.com indicando:
- Nombre
- Apellidos
- Dirección Completa
- Población
- Provincia
- Cógigo Postal
- Mail de Contacto y/o Teléfono Contacto
Es imprescindible que nos facilites un mail o teléfono de contacto.
- Tipo de Subscripción: CONTRAREEMBOLSO
- Número de Revista:
Este será el número a partir del cual quieres subscribirte. Si deseas (por ejemplo) subscribirte a partir del número 5 (incluido), debes poner un 5 y te enviaremos desde el 5 hasta el 15 (ambos incluidos)

APRECIACIONES:
* Junto con el primer número recibirás el abono de 45 euros, precio de la subscripción por 11 números (un año) y una carta donde se te indicará tu número de Cliente Preferente y justificante/factura de la subscripción.
* Puedes hacernos llegar estos datos POR MAIL,tal como te hemos indicado; rellenando el formulario de nuestra WEB (www.hackxcrack.com) o enviándonos una carta a la siguiente dirección:
CALLE PERE MARTELL Nº20, 2º-1ª
CP 43001 TARRAGONA
ESPAÑA
* Cualquier consulta referente a las subscripciones puedes enviarla por mail a preferente@hackxcrack.com

Giro Postal

Envíanos un GIRO POSTAL por valor de 45 EUROS a:
CALLE PERE MARTELL20, 2º 1ª.
CP 43001 TARRAGONA
ESPAÑA
IMPORTANTE: En el TEXTO DEL GIRO escribe un mail de contacto o un número de Teléfono.

Y enviarnos un mail a preferente@hackxcrack.com indicando:
- Nombre
- Apellidos
- Dirección Completa
- Población
- Provincia
- Cógigo Postal
- Mail de Contacto y/o Teléfono Contacto
Es imprescindible que nos facilites un mail o teléfono de contacto.
- Tipo de Subscripción: GIRO POSTAL
- Número de Revista:
Este será el número a partir del cual quieres subscribirte. Si deseas (por ejemplo) subscribirte a partir del número 5 (incluido), debes poner un 5 y te enviaremos desde el 5 hasta el 15 (ambos incluidos)

APRECIACIONES:
* Junto con el primer número recibirás una carta donde se te indicará tu número de Cliente Preferente y justificante/factura de la subscripción.
* Puedes hacernos llegar estos datos POR MAIL,tal como te hemos indicado; o enviándonos una carta a la siguiente dirección:
CALLE PERE MARTELL Nº20, 2º-1ª
CP 43001 TARRAGONA
ESPAÑA
* Cualquier consulta referente a las subscripciones puedes enviarla por mail a preferente@hackxcrack.com

EL LENGUAJE XSL

TRANSFORMACION DEL DOCUMENTO XML

POR JOAQUIM ROCA VERGES

El curso XML provocó muchos mails en los que se repetía la misma pregunta... ¿Para qué sirve realmente? ¿Cómo puedo utilizarlo para mi Web? ¿Cómo utilizo la información? LA RESPUESTA ES XSL -----> PREPÁRATE!!!

Para la comprensión y asimilación de esta serie de artículos sobre XSL, entendemos que se tienen conocimientos básicos de HTML, y que se han leído y comprendido los artículos de la serie XML publicados en PC Paso a Paso del nº 10 al 16.

Si no habéis leído estos artículos os será muy difícil entender XSL, os recomiendo que entendáis y asimiléis como mínimo el artículo de la revista 10 (básico e indispensable), la teoría del DOM en la revista 13 (básico e indispensable) y a ser posible el núcleo del DOM en la revista 14 (recomendable pero no indispensable).

Si no sabéis mucho o nada de HTML, miraros algún texto de introducción de teoría HTML y los TAGS básicos:

- <HTML>
- <TITLE>
- <BODY>
- <TABLE> – <TH> - <TR> – <TD>
- <A>
- <P>
- <IMG>

Ayuda mucho entender la teoría antes de ponernos a practicar. Si uno "pierde tiempo" en entender la teoría, y realmente lo logra, luego os puedo asegurar que creará programas como una bala, de manera intuitiva y casi sin la ayuda.

Adelante.

El Lenguaje XSL (e**X**tensible **S**tylesheet **L**anguage) define una serie de reglas que especifican como extraer la información de un documento XML y como formatear esta información de manera que pueda ser visualizada.

XSL se divide en tres partes:

- **XSL Transformations (XSLT)** = Es un lenguaje para transformar documentos XML
- **XML Path (Xpath)** = El SQL de XML, un lenguaje de consulta sobre los elementos del documento XML, un lenguaje para referenciar y seleccionar partes de un documento.
- **XML Formating Objects (XSL – FO)** = Un vocabulario XML utilizado por XSLT que pinta el texto formateado en una página.

XSLT

XSLT es un lenguaje que sirve para transformar documentos XML. Usamos XSLT para especificar la manera en que un documento XML será transformado en otro tipo de documento. Aunque el documento resultante de transformar un XML con XSL puede ser otro documento XML, un PDF, un archivo JSP, ASP, JAVA..., lo usual es que el producto de esta transformación sea un documento HTML, de modo que los usuarios puedan verlo.

El proceso de transformación "típico" es el siguiente:

- Un documento de entrada(xml) se empareja con un conjunto de uno o mas documentos XSLT llamados hojas de estilo XSLT (XSLT StyleSheets).

- En las hojas de estilo XSLT escribimos las reglas de transformación.

- Durante el proceso de transformación , un procesador XSLT analiza el contenido del documento de entrada (el XML) para emparejarlo con las reglas del XSL. Estas reglas, de transformación están organizadas como plantillas(templates) que definen acciones que se tienen que tomar cuando se encuentra la pareja del XSL en el XML.

Por ejemplo, en un xml tenemos un elemento llamado "introduccion"

```
<introduccion> XSL es una herramienta de transformación potente</introduccion>
```

y en el XSL tenemos:

```
<xsl:template match="introduccion">
```

Podemos observar que el elemento <introduccion> casa (empareja, forma el par) con el valor de la propiedad match del xsl..

Cuando el procesador XSLT determina que un elemento del XML tiene pareja en el XSL, escribe su contenido a un buffer (espacio de memoria temporal , espacio de almacenamiento de información temporal) de salida. Una vez ha acabado el análisis del documento XML, el procesador XSL puede reestructurar el buffer de salida (que es donde están los datos del XML analizado) para formar un documento HTML por ejemplo.

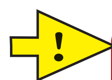
Que podemos hacer con un xsl:

1. **Dar formato a un documento** cambiar las fuentes, márgenes, tamaño de la letra, etc.
2. **Ordenar** Podemos cambiar el orden de los elementos, de acuerdo con un criterio. Por ejemplo en un XML con datos bancarios , con XSL podemos ordenar según el número de cuenta.
3. **Filtrar** Podemos eliminar elementos que no necesitamos, mostrando solo los que cumplen las reglas que deseamos. Por ejemplo: en un XML con datos de ordenes de compra podemos filtrar las ordenes de compra no terminadas de las finalizadas.
4. **Calcular** Podemos realizar cálculos aritméticos. Por ejemplo en un XML con datos de ventas, podemos calcular la suma total del dinero que tenemos que cobrar.
5. **Anexar, combinar** Podemos combinar múltiples documentos en uno solo. Por ejemplo en varios XML con datos de ventas, podemos combinarlos todos en un solo documento de resumen llamado "Ventas mensuales".

Cuando tengas que hacer cambios de XML a otro formato, o tengas que realizar alguna de las cinco operaciones que he especificado sobre documentos XML, no lo dudes: XSL es la herramienta que debes elegir.

Es esencial que el proceso de transformación se entienda a la perfección para trabajar con XSLT, por lo tanto, vamos a verlo más detalladamente.

Empezaremos con la creación de un archivo de entrada XML.



Un archivo de...

Un archivo de entrada es un archivo del cual leemos la información que procesaremos.

Escribimos en cualquier editor de texto:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<introduccion>
XSL es una herramienta de transformación potente
</introduccion>
```

y guardamos el archivo como introduccion.xml en una carpeta que denominaremos xsl, y colgará del directorio raíz de Windows: c:\xsl

Aunque podemos ver el documento XML directamente en el browser de Internet (el Internet Explorer de Windows, el antiguo Navigator de Netscape, etc.), el documento solo es una estructura de datos bien formada, pero no podemos visualizar la información formateada.

Para ello, para formatear la información y visualizarla , transformaremos el documento a otro formato, a HTML por ejemplo, y ello lo vamos a hacer con XSLT, con una hoja de estilos XSLT (xslt stylesheet) donde especificaremos como vamos a transformar el documento XML "introduccion xml".

De momento ya tenemos el XML que vamos a transformar, ahora vamos a decirle como vamos a transformarlo , y esto vamos a hacerlo con un archivo XSL

Abrid cualquier editor de texto y escribid lo siguiente:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>

<xsl:template match="/">
<xsl:apply-templates select="introduccion"/>
</xsl:template>

<xsl:template match="introduccion">

<html>
        <body>
        <p>
                <xsl:value-of select="."/>
        </p>
        </body>
</html>

</xsl:template>

</xsl:stylesheet>
```

Una vez lo tengas escrito, guarda el archivo en la misma carpeta xsl con el nombre: "introduccion.xsl"

¿En que nos podemos fijar de entrada? Pues en que el documento está bien formado. Recordamos que un documento bien formado es el que cumple las reglas de escritura de XML. Por ejemplo, todo elemento que se abre, a no ser que se especifique lo contrario, debe cerrarse:

Se abre: <xsl:template>
Se cierra:</xsl:template>

A veces se especifica lo contrario: <xsl:output method="html"/> ved que el slash"/" está al final, lo que indica que el elemento es único, que se cierra y abre el mismo.

Por tanto debemos tener en cuenta que la manera de escribir, sea XSL, XML , HTML o cualquier otro lenguaje de marcado, debe terminar generando un archivo bien formado, esto es que sigue las reglas del XML.

Si alguno de vosotros tiene conocimientos de HTML, puede ver que la etiqueta "<P>" (que cuando escribes un HTML puede dejarse sin cerrar) aquí la hemos cerrado con una etiqueta fin de párrafo "</P>"

Aunque dentro de nada vamos a examinar en detalle las hojas de estilo XSLT, vamos a prestar atención a los fundamentos.

La primera plantilla(template) de la hoja de estilo XSLT, dice al procesador XSLT que encuentre el elemento raiz(root) :

```
<xsl:template match="/">
```

La segunda plantilla(template) :

```
<xsl:template match="introduccion">
```

Reemplaza el tag de principio <introduccion> y el tag de final </introduccion> por el tag HTML que le hemos indicado en el XSL: <p> y después inserta el valor del elemento <introducción> al tag <p> del html.

El resultado final es el documento HTML:

```
<html>
        <body>
        <p>
          XSL es una herramienta de transformación potente
        </p>
        </body>
</html>
```

Vamos a ponerlo en práctica.

Tenemos por un lado el XML que tiene la información que vamos a procesar, y por otro lado el XSL que tiene las reglas de lo que va a hacer el procesador.

Necesitamos pues el procesador, la aplicación que va a convertir el XML, siguiendo las reglas del XSL en una página Web HTML.

Como es una parte muy importante, vamos a dar dos soluciones (si es posible, os recomendaría tuvierais las dos).

1) Software Libre.
2) Software de pago pero…

1.-Software Libre

Xalan del **Apache XML Project**
http://xml.apache.org/xalan-j/index.html
(aquí encontrareis toda la información que queráis acerca del xalan y mas).

Este es quizá el procesador XSLT más popular que existe. Para que funcione tienes que instalarte el java development kit (JDK) de http://java.sun.com/j2se/1.4.1/download.html

tambien tienes que bajarte el j2ee (java 2 enterprise edition) de

http://java.sun.com/j2ee/sdk_1.3/

Una vez instalado todo (jdk + j2ee) comprobad que en tenéis en vuestra estructura de directorios la carpeta C:\j2sdkee1.3\lib\org\apache\xalan, en caso de que no la tengáis, buscad el archivo j2ee.jar y descomprimidlo dentro de la carpeta c:\ j2sdkee1.3\lib .

En la revista PC Paso a Paso nº 12 tenéis las instrucciones paso a paso de cómo hacer la instalación del java y de la configuración del CLASSPATH.

Una vez tengáis instalado correctamente el j2ee, abrimos una sesión del intérprete de comandos y tecleamos:

java org.apache.xalan.xslt.Process –in archivoXML –xsl archivoXSL –out archivoDeSalidaHTML

sustituyendo archivoXML por introduccion.xml, archivoXSL por introduccion.xsl y archivoDeSalidaHTML por introduccion.html:

```
C:\xsl>java org.apache.xalan.xslt.Process -in introduccion.xml -xsl introduccion
.xsl -out introduccion.html
```

y esto os provocará el siguiente error:

```
C:\xsl>java org.apache.xalan.xslt.Process -in introduccion.xml -xsl introduccion
.xsl -out introduccion.html
Exception in thread "main" java.lang.ClassCastException: org.apache.xalan.res.XS
LTErrorResources_es
        at org.apache.xalan.xslt.Process.main(Process.java:209)
```

Bueno pues nada, a coger moral y a investigar porque de este error. Veamos, seguramente necesitamos mas archivos de funciones. Nos vamos a la Web de apache y nos bajamos el xalan:

http://xml.apache.org/dist/xalan-j/xalan-j_2_4_1-bin.zip

Copiad el zip a vuestro directorio raíz: "C:\", seleccionadlo y buscad la opción del WinZip que pone "Extract Here" de manera que una vez descomprimido, en vuestra estructura de directorios veáis la carpeta C:\ xalan-j_2_4_1.

Configuración del CLASSPATH

Para hacer una clase accesible a una aplicación Java, la aplicación debe saber donde encontrarla. En la variable de entorno CLASSPATH, se informa al ejecutable java donde encontrar las clases.

Añadir al Classpath las siguientes entradas:

- C:\ xalan-j_2_4_1\bin\xalan.jar
- C:\ xalan-j_2_4_1\bin\xercesImpl.jar
- C:\ xalan-j_2_4_1\bin\xml-apis.jar

Y reiniciad la máquina. Una vez hemos reiniciado, comprobad que en el CLASSPATH, existen las tres entradas que hemos añadido.

A continuación vamos a probar que tenemos correctamente instalado el xalan, vamos a compilar una clase java de los ejemplos de xalan, que tenéis en la carpeta

C:\xalan-j_2_4_1\samples\SimpleTransform

Abrid una sesión del intérprete de comandos y situaros en la carpeta SimpleTransform y teclead:

javac SimpleTransform.java

si no os da ningún error teclead a continuación:

java SimpleTransform

y si tenéis el Xalan correctamente instalado os saldrá el siguiente mensaje

************* The result is in birds.out *************

Lo veis en la siguiente imagen:

```
C:\>cd C:\xalan-j_2_4_1\samples\SimpleTransform
C:\xalan-j_2_4_1\samples\SimpleTransform>javac SimpleTransform.java
C:\xalan-j_2_4_1\samples\SimpleTransform>java SimpleTransform
************* The result is in birds.out *************
C:\xalan-j_2_4_1\samples\SimpleTransform>
```

Vamos a probar con nuestro ejemplo: Recordamos que tenemos:

1. archivo de entrada (IN) de donde vamos a leer los datos, que es un XML llamado introduccion.xml
2. archivo de transformacion (XSL) con el cual vamos a aplicar las transformaciones al archivo de entrada

Y finalmente haremos que se genere como archivo de salida (OUT) una página HTML, que llamaremos introduccion.html

Para ello, abriremos una sesión del intérprete de comandos, nos situaremos en la carpeta C:\xsl y escribiremos:

java org.apache.xalan.xslt.Process –IN introduccion.xml –XSL introduccion.xsl –OUT introduccion.html

y si abrimos la carpeta xsl, veremos un nuevo archivo, un flamante html, que hemos generado con XSLT.

En la imagen siguiente vemos el archivo XML original de entrada y el HTML de salida que ha pasado el proceso de transformación XSL:



2. Software De pago pero...

Indudablemente si tenéis acceso a software de pago, esta es la opción mas sencilla, no tendréis que configurar nada, con unos pocos pasos tendréis el archivo generado.

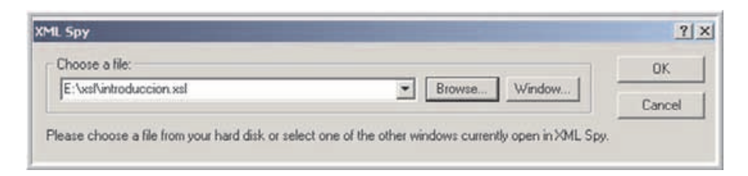
Os recomiendo XML Spy, que podéis bajaros de *www.altova.com/download_archive.html*

Supongo que entendéis lo del pero..., por la Web corren muchos peros... Yo utilizo la versión 4.4 de xml spy, que la podéis encontrar en :

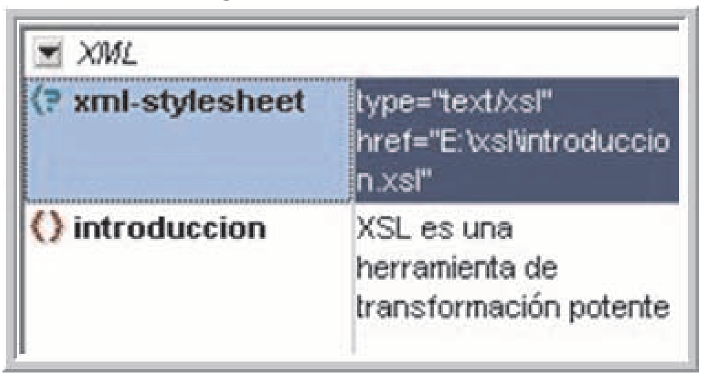
www.altova.com/download_archive.html con el nombre de XML Spy 4.4 Suite Setup.

Una vez tengáis instalado el xml spy, abrid el archivo introduccion.xml y seleccionad la opción de menú **XSL – Assign XSL**

Haced click en el botón **Browse...** seleccionad el archivo introduccion.xsl y haz click en el botón **Ok:**



Podéis comprobar que el xml spy os ha añadido código extra a vuestro archivo XML



Una vez habéis asignado el XSL al XML, para transformarlo en un html seleccionad la opción de menú **XSL – XSL Transformation,** os saldrá una nueva ventana con el archivo html generado. Solo tenéis que guardarlo como introduccion.html.

Cuando trabajéis con XSLT, debéis mentalizaros que el único template que se aplica automáticamente a cualquier documento, es el template que se aplica al elemento raiz.
Recordamos que los templates lo que hacen es buscar el elemento que especificamos en el xsl, en el xml. Para todos los demás templates, deberéis ir especificando que el documento xsl los busque en el xml. Esto lo haremos definiendo un template raiz que uno detrás de otro va llamando otros templates.

Los templates siempre se procesan uno detrás de otro, así si un template tiene templates anidados, se procesaran todos sus templates anidados, y cuando finalice su proceso se pasará a procesar el siguiente template. Lo vemos con un ejemplo:

```
<xsl:stylesheet
        xmlns:xsl=http://www.w3.org/1999/XSL/Transform
        version="1.0"

        <xsl:template match="/">
        <xsl:apply-templates select="elemento1" />
        <xsl:apply-templates select="elemento2" />
        <xsl:apply-templates select="elemento3" />
        </xsl:template>
....
....
</xsl:stylesheet>
```

En este ejemplo, el template raíz

```
        <xsl:template match="/">
```

especifica que hay templates para tres elementos:

```
        <xsl:apply-templates select="elemento1" />
        <xsl:apply-templates select="elemento2" />
        <xsl:apply-templates select="elemento3" />
```

El procesador XSL comenzará con el template raíz y después irá a procesar el template para el elemento1. Si el elemento1 tiene templates anidados o simplemente las reglas que le hemos puesto llaman a otros templates, los templates que están dentro del elemento1 serán procesados ordenadamente uno tras otro y solo cuando los templates que están dentro del elemento1 hayan sido procesados, pasaremos al elemento2, luego al elemento3 y así hasta finalizar.

ESTRUCTURAS DE DOCUMENTO XML QUE SE EMPAREJAN CON LAS REGLAS DE LOS TEMPLATES

La recursión o recursividad es un aspecto muy importante en XSLT. Cuando los procesadores XSLT analizan los documentos de entrada (los XML), los ven como árboles de nodos, donde cada uno de los nodos representan partes individuales del documento XML como pueden ser los elementos, los atributos, el texto etc., y el árbol de nodos representa el documento XML por entero.

Arriba del todo, en la parte mas alta del árbol de nodos está el nodo raíz, que representa el elemento raíz del

documento XML, y por debajo los nodos hijos(nodos anidados) del raíz, que a su vez pueden tener hijos (mas nodos anidados) y también pueden tener nodos hermanos (nodos que están a un mismo nivel).

Pensad en la estructura de directorios:

- C:\ sería el nodo raíz
- C:\Archivos de Programa seria un nodo hijo de C:\
- C:\Archivos de programa\Office, seria un nodo hijo de C:\Archivos de Programa
- C:\Windows sería un nodo hermano de C:\Archivos de Programa.

Esta estructura en árbol hace la tarea fácil al procesador XSL que entiende a la perfección la estructura PADRE-HIJO-HERMANO.

La tecnología que hace posible que la estructura del documento XML se represente en una estructura de árbol se llama **Xpath.** Xpath define un conjunto de nodos estándar y nos da las funciones para localizar estos nodos en el documento XML.

Tipos nodo definidos por Xpath:

- **Raiz** = representa el elemento raíz de los documentos XML. El nodo raíz contiene todo el documento y todos los nodos del documento son hijos suyos.
- **Elemento =** representa elementos de los documentos XML, incluyendo el elemento raíz. Los nodos elemento, pueden incluir otros nodos elemento, nodos de texto, nodos de comentario e instrucciones para procesar que se desarrollan dentro del elemento.
- **Atributo** = representa elementos atributos en los documentos XML. Los atributos no son ni padres, ni hermanos, ni hijos de ningún nodo.
- **Texto =** representa el texto contenido en los elementos. Si el texto asociado a un elemento contiene referencias externas a entidades (por ejemplo una imagen) o tipos de caracteres (signos raros de diferentes abecedarios por ejemplo), estas referencias se resuelven (se trae el contenido de las referencias) antes que el nodo texto sea creado.
- **Comentario =** representa elementos comentario en los documentos XML.
- **Instrucción para procesar =** representa instrucciones que hay que procesar en los documentos XML. Los nodos de este tipo contienen dos valores: el nombre de la instrucción que puede obtenerse llamando a la **función name( )** y una cadena de tipo String con la instrucción en sí.
- **Namespace =** representan namespaces declarados en hojas de estilo XSLT. Un namespace se utiliza para prevenir conflictos de nombres dentro de un mismo XML. Supongamos que utilicemos la misma etiqueta <titulo> para designar diferentes conceptos dentro del XML. Por ejemplo, podemos tener un <titulo> que nos da el nombre de un libro, y dentro del mismo documento XML, otro <titulo> que nos da el título obtenido después de años de estudio. Este problema se resuelve con los namespaces.

Cada uno de estos tipos de nodo tiene una plantilla(template) asociada, que permite que el nodo sea procesado por XSLT cuando sea necesario. Vamos a examinar las reglas de cada uno de estos templates en el próximo número, pero antes vamos a ver dos ejemplos sencillos, porque me gustaría que antes de

saber las reglas, por intuición, simplemente leyendo el código os imaginéis lo que va a ocurrir.

Interesaría que viendo el XML y las reglas XSL que vamos a aplicar, pudiésemos intuir de antemano el archivo de salida HTML. También sería igual de interesante que, viendo la salida HTML y volviendo a leer el código, entendiésemos lo que ha ocurrido.

Supongamos que somos de un equipo de estadística y se nos pide el número de nacimientos que ha tenido la ciudad de Martorell, durante la década de los años 90, del siglo pasado.

Escribimos en cualquier editor de texto:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<infonatalicios>
        <natalicios ciudad="Martorell" numNacimientos="489" anyo="1990"/>
        <natalicios ciudad="Martorell" numNacimientos="587" anyo="1991"/>
        <natalicios ciudad="Martorell" numNacimientos="685" anyo="1992"/>
        <natalicios ciudad="Martorell" numNacimientos="324" anyo="1993"/>
        <natalicios ciudad="Martorell" numNacimientos="285" anyo="1994"/>
        <natalicios ciudad="Martorell" numNacimientos="999" anyo="1995"/>
        <natalicios ciudad="Martorell" numNacimientos="125" anyo="1996"/>
        <natalicios ciudad="Martorell" numNacimientos="1045" anyo="1997"/>
        <natalicios ciudad="Martorell" numNacimientos="839" anyo="1998"/>
        <natalicios ciudad="Martorell" numNacimientos="450" anyo="1999"/>
</infonatalicios>
```

y guardamos el archivo como nacimientos.xml en la carpeta xsl: c:\xsl\nacimientos.xml

Abrid cualquier editor de texto y escribid lo siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="/">
  <html>
    <head>
      <title>Nacimientos en la década de los años 90</title>
    </head>
    <body>

      <h1>
      Nacimientos en la década de los años 90
      </h1>

      <p>Número de nacimientos en la ciudad de Martorell durante
           la decada de los 90
      </p>
      <table border="1">

           <tr>
                <th>Año</th>
                <th>Num nacimientos</th>
                </tr>

           <xsl:for-each select="infonatalicios/natalicios">
           <tr>
                <td>
                <xsl:value-of select="@anyo"/>
                </td>
                <td>
                <xsl:value-of select="@numNacimientos"/>
                </td>
           </tr>
           </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

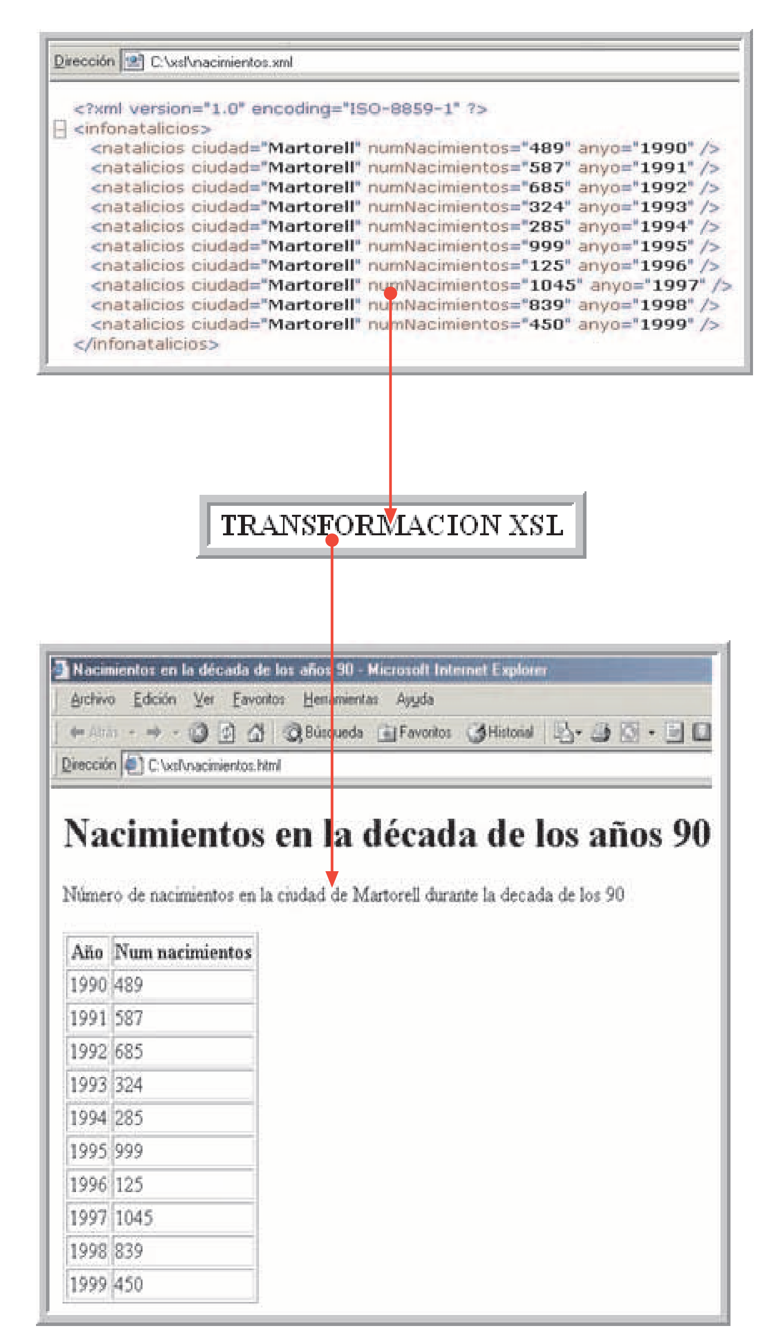
Una vez escrito guardamos el archivo en la misma carpeta xsl con el nombre: “nacimientos.xsl”

Ahora nos falta convertir el xml en html, con el procesador XSL. Para ello, abriremos una sesión del intérprete de comandos, nos situaremos en la carpeta C:\xsl y escribiremos:

java org.apache.xalan.xslt.Process –IN nacimientos.xml –XSL nacimientos.xsl –OUT nacimientos.html

Y si abrimos la carpeta XSL, veremos un nuevo archivo, el HTML que hemos generado con XSLT.

En la imagen siguiente vemos el archivo XML original de entrada y el HTML de salida que ha pasado el proceso de transformación XSL:



El segundo ejemplo es una satisfacción personal, le tenía ganas desde que un compañero me lo preguntó en el foro de la Web de la revista (http://www.hackxcrack.com/)

Se trata de convertir un archivo XML con un listado de películas en links HTML.

Abrid un editor de texto y escribid:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xvid>
<accion>
        <titulo http="http://whatisthematrix.warnerbros.com/">Matrix</titulo>
        <titulo http="http://www.xmen2lapelicula.com/">X-Men2</titulo>
        <titulo http="http://www.apple.com/trailers/fox/daredevil/">Daredevil</titulo>
        <titulo http="http://www.thehulk.com/index_flash.html">Hulk</titulo>
</accion>
</xvid>
```

Y guardamos el archivo como videos.xml en la carpeta xsl: c:\xsl\videos.xml

Abrid cualquier editor de texto y escribid lo siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="/">
  <html>
    <head>
    <title>Convertir un XML en un HTML</title>
    </head>
    <body>
      <h1>Videos de acción</h1>
      <p>Listado de videos</p>
      <table border="0">
            <tr>
                  <th>Video</th>

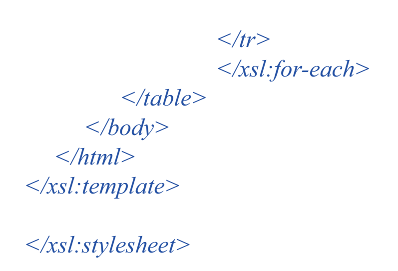
                  <th>Dirección http</th>

                  <th>Nombre convertido en link </th>
            </tr>
            <xsl:for-each select="xvid/accion/titulo">
            <tr>
                  <td>
                  <xsl:value-of select="."/>
                  </td>

                  <td>
                  <xsl:value-of select="@http"/>
                  </td>

                  <td>
                  <a>
                  <xsl:attribute  name="href">
                  <xsl:value-of select="@http"/>
                  </xsl:attribute>

                  <xsl:value-of select="."/>
                   </a>
                  </td>
```



```
                </tr>
                </xsl:for-each>
        </table>
      </body>
    </html>
</xsl:template>

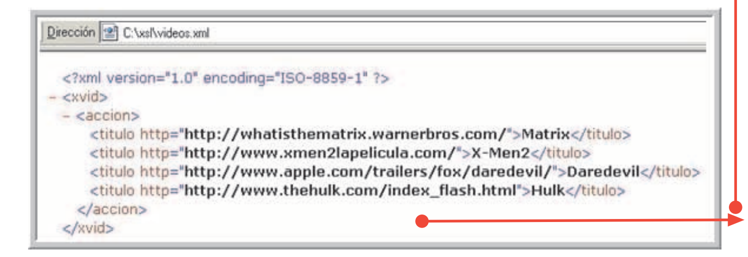
</xsl:stylesheet>
```

Abrimos una sesión del intérprete de comandos, nos situamos en la carpeta xsl y escribimos:

java org.apache.xalan.xslt.Process –IN videos.xml –XSL videos.xsl –OUT videos.html

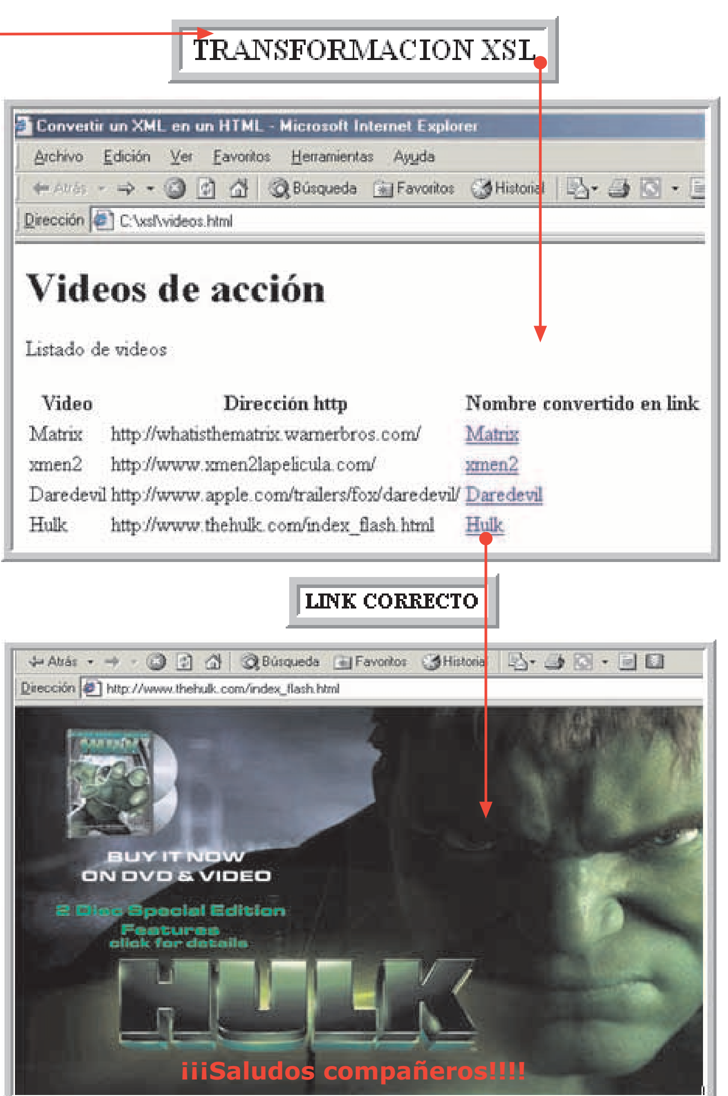
y si abrimos la carpeta xsl, veremos un nuevo archivo, el HTML que hemos generado con XSLT.

En la imagen siguiente vemos el archivo XML original de entrada y el HTML de salida que ha pasado el proceso de transformación XSL y verificamos que los links son correctos:



Dirección C:\xsl\videos.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <xvid>
  - <accion>
      <titulo http="http://whatisthematrix.warnerbros.com/">Matrix</titulo>
      <titulo http="http://www.xmen2lapelicula.com/">X-Men2</titulo>
      <titulo http="http://www.apple.com/trailers/fox/daredevil/">Daredevil</titulo>
      <titulo http="http://www.thehulk.com/index_flash.html">Hulk</titulo>
    </accion>
  </xvid>
```



LA REALIDAD DE LOS VIRUS INFORMATICOS

- ¿Qué son los Virus Informáticos? - Tipos de Infecciones - Tecnicas de Ocultamiento
- Un poco de CODIGO MAQUINA - Heurística - Probando nuestro antivirus
- Estructura de los ficheros Ejecutables - Fakes, Jokes y Hoaxes

1- La realidad de los virus informáticos

Hace ya bastante tiempo durante la época de los 60, en los conocidos "Bell Labs" de AT&T, los investigadores de inteligencia artificial desarrollaron lo que comenzó siendo un curioso juego al que llamaron "Core Wars" y que se basaba fundamentalmente en dos programas que "luchaban" entre ellos intentando eliminarse mutuamente.

Nada más lejos de la realidad, estos programas con "instinto de supervivencia" se crearon en un lenguaje pseudo-ensamblador llamado RedCode y se ejecutaban gracias al 'Memory Array RedCode Simulator' o MARS.

En 1984 se define públicamente el término "virus" aplicado a la informática por parte de Fred Cohen como "un programa dañino con capacidad para "auto-replicarse" al mismo tiempo que hacía inevitables referencias a los evidentes paralelismos con los virus de carácter biológico.

Actualmente, en la era de la información se difunden todo tipo de textos sobre virus informáticos sin ningún tipo de base técnica, al más puro estilo sensacionalista, lo que únicamente contribuye a causar un miedo irracional entre los usuarios.

Pero en cualquier caso, los virus suponen actualmente un problema real tanto a nivel de usuario como empresarial, lo que nos lleva a preguntarnos ¿Qué es un virus informático? o mejor aún ¿Qué no es un virus informático?

- En primer lugar, lo que debe quedar muy claro es que un virus informático es solo un programa de ordenador – elaborado y preciso - pero un programa como cualquier otro al fin y al cabo (la mayoría programados en lenguaje ensamblador por dar un resultado más óptimo, pero los hay en otros lenguajes de programación como C), por tanto, podemos ver la "creación de virus" como "programación de código auto-replicante" para una definición más correcta.

- Como acabo de decir, disponen de capacidades de auto-replica mediante diversas técnicas que veremos luego paso a paso ;-)

- Aunque en muchos casos ocasionan daños o puedan explotar algún agujero de seguridad del sistema, no siempre es así, de hecho los hay que se limitan a duplicarse por el sistema consumiendo recursos sin más o infectándose a velocidades increíbles en todo tipo de plataformas y ficheros.

- Suele pasar desapercibido al usuario y a los sistemas de defensa del sistema para ser efectivo y poder duplicarse.

- Un virus informático se duplica inyectando su código en un objeto o programa ejecutable adecuado de manera que al ejecutar dicho objeto el virus toma el control total y hará la tarea que tiene programada (se ejecuta el virus) y devuelve el control de forma transparente al programa infectado.

- Muchos habrán escuchado aquello de "en Linux no hay virus", mentira, si que los hay aunque en mucha menor proporción y con un impacto bastante limitado por motivos que se tratarán a continuación. De todas maneras sí es cierto que bajo Linux no tenemos que preocuparnos en exceso por una posible infección.

¿Que no puede hacer un virus?

- Un virus no puede escribir en memoria ROM (como en los CD-ROM) o discos protegidos contra escritura.

- No podemos infectarnos por ver una imagen tipo "*.jpg","*.gif" o un "*.bmp",etc. Las imágenes son simplemente eso –imágenes- y no se ejecutan sino que se interpretan por programas gráficos.

- No se puede infectar un fichero sin manipularlo y en la mayoría de los casos (por no decir todos) modificarlo.

- Todos los virus pueden ser detectados, ¡ojo! me refiero a que no hay ningún virus que nunca pueda ser detectado por un sistema antivirus.

- No hay ningún virus que no pueda eliminarse ya que simplemente son secuencias de bytes. La única excepción puntual podría ser el caso de los virus de sobre-escritura en los que se elimina parte del fichero original o si el virus se encuentra en un soporte de solo lectura como un CD-ROM por ejemplo.

- Las "cookies" no contienen virus, son ficheros de texto plano y los ficheros de texto no te pueden infectar, no son ejecutables.

- El software "legal" también puede contener virus, puedes infectarte tanto si el software que utilizas es legal como si usas "copias pirata".

- Un virus no puede dañar el hardware... mmm, a ver, un virus puede llegar **en los peores casos a dejar inservible el hardware pero no dañarlo físicamente...** si un virus hace que tu disco duro escriba y borre muy rápido y muy seguido en un mismo sector durante horas... quizás, si se te "borra la Flash BIOS" del ordenador... quizás, aunque posible, esto es por lo general poco probable.

Así, podemos afirmar que un virus debe ejecutarse para poder infectar o hacer aquello para lo que fue programado y por tanto no podemos ser infectados por visitar una Web o leer un mensaje de texto, ver una imagen, etc.

Bueno jeje, esto no es totalmente cierto. Actualmente los sistemas y programas de la compañía Micro$oft (entre otros) tienen numerosos agujeros de seguridad que permiten la ejecución de código automáticamente sin consentimiento del usuario, es el momento de mencionar el repertorio de fallos en el programa Outlook y las recientes vulnerabilidades en M$ Internet Explorer xD, por supuesto ya deberías tener aplicados los parches correspondientes para evitar que esas u otras aplicaciones puedan dar lugar a 'una' infección ;-) es importante actualizarse y no dejar nuestro sistema "como Bill lo trajo al mundo" .

Recientemente con la propagación del gusano 'Blaster' se tiene cada vez más presente :P. Si piensas que proteger el equipo contra los temidos virus es imposible evidentemente te equivocas, ya que siguiendo una serie de pautas fundamentales (muchas de sentido común) es realmente poco probable que infectes tu sistema, recuerda: es mejor prevenir que curar.

¿Cómo prevenirnos?

De eso y otras cosas te hablaré ahora en este artículo, la mejor manera de enfrentarnos a este tipo de programas auto-replicantes es

conocer su funcionamiento… quien sabe, quizás les cojas cariño a estas criaturas y decidas dar a luz una tu mismo algún día, nunca se sabe :P

2- Nociones fundamentales

Antes de adentrarnos un poco más en el curioso mundo de estas criaturas para conocer las distintas especies que podemos encontrar, nunca va mal hacer un breve repaso de algunos conceptos acerca de la arquitectura de computadores.

Una computadora, desde un punto de vista simplista, se compone de:

C.P.U (Unidad Central de Procesamiento): su función es determinar como se ejecuta el código, como mandar los datos y generar otras señales de control para el sistema.

Memoria: sirve como medio de almacenamiento de información en la computadora con un acceso de velocidad medio comparando el tiempo de acceso al de los registros de la CPU, la memoria caché y un disco duro, siendo el último el medio más lento pero el que tiene más capacidad de almacenamiento.

Dispositivos I/O (entrada y salida): nos referimos a todos los periféricos como por ejemplo el teclado, el módem, tarjetas de sonido, etc. que mandan o reciben datos de la máquina.

Bus de comunicaciones: representan flujos de datos dentro del computador y diversas partes que lo componen.

Para todo lo referente a la realización de operaciones aritméticas y lógicas tenemos la ALU (Unidad Aritmético-Lógica). Como operaciones aritméticas tenemos la suma y la resta, el incremento, decremento y quizás multiplicación y división. Como operaciones lógicas básicas tenemos:

- AND, devuelve 1 si los dos bits son 1: *1001 AND 1000* = 1000 (se corresponde con la multiplicación binaria).
- OR, devuelve 1 si cualquiera de los bits es 1: *1001 OR 1000* = 1001 (se corresponde con la suma binaria).
- NOT, invierte los valores de los bits: *NOT (1001)* = 0110

También existe una unidad de registros de la CPU que resulta fundamental para no tener siempre que acceder a la memoria y poder trabajar en un medio más rápido en cuanto a tiempo de acceso. Su uso es diverso: ejecutar instrucciones, guardar datos, etc.

Existen varios tipos de registros para **uso general** (visibles al programador) y **específicos** (reservados para el sistema y no se pueden usar directamente), los primeros podrían ser EAX, EDX, ECX, EDI, EBP y otros tantos, ejemplo de los segundos serían -entre otros- el registro SP o "Stack Pointer" , el MAR ("Memory Address Register") o el PC ("Program Counter") cada uno con funciones muy concretas. Por ejemplo, el PC es el contador de programa y es el que contiene la siguiente dirección a la que se ejecuta en cada instante.

2.1 Nociones generales sobre los sistemas operativos

El sistema operativo nos supone una interfaz con la que trabajar con nuestra computadora y nos brinda una serie de funciones elementales (y no tan elementales) encargándose de:

- gestionar los recursos del ordenador
- ejecutar nuestras órdenes
- y ofrecer servicios a los programas. Por ejemplo servicios que permiten su ejecución, abrir un fichero y el manejo de excepciones (errores y demás) entre otras muchas cosas.

Se les llama nivel de Kernel, Shell y API respectivamente.

2.1.2 El arranque del computador

Este resulta siempre un punto interesante, muchos no saben qué es lo que ocurre cuando nosotros tan felizmente pulsamos el botón de "power on" y esperamos con cara de satisfacción el despertar de nuestro inseparable amigo (--¡y no me refiero al Window$! :P). Pues ahora te lo voy a explicar paso a paso para que veas que las cosas simples son las más importantes:

El primer paso es ejecutar lo que se llama "ROM Starter" o iniciador ROM. Es un programa que siempre se encuentra cargado (en memoria de sólo lectura y/o lectura exclusiva) y que comprueba el sistema y los periféricos conectados.

A continuación enviará a memoria otro programa que se responsabilizará de cargar el sistema operativo. En el caso de tener nuestro disco duro (hoy día de tamaños considerables) lo que cargaremos será el MBR ("Master Boot Record", es el primer sector del disco duro que contiene la tabla de particiones).

Desde aquí se da el control al primer sector o "boot sector" que son esos 512 bytes iniciales que ya pueden cargar el resto del programa y una vez habilitados los componentes se inicializa el Sistema Operativo (para el caso que nos ocupa será Window$) y que realizará - para ir bien - un test del sistema (completando así la tarea iniciada en la ROM), comprobación de integridad del sistema de ficheros, establecerá sus estructuras y políticas propias (como la planificación de procesos y la memoria) y pondrá residentes en la memoria aquellas partes vitales del SO. Ahora simplemente se crea un proceso inicial o proceso padre que incluye una petición y comprobación de usuario / contraseña opcionalmente.

Llegados a este punto podemos trabajar con las posibilidades que el SO nos ofrece.

2.1.3 Procesos

Dejando la filosofía a un lado podemos decir que un proceso es un programa que está en ejecución y es la unidad de procesamiento que el SO debe gestionar correctamente. La información sobre cada proceso se guarda en su mayoría en el BCP (Bloque de Control de Proceso) que tiene información de interés como podría ser:

- UID: identificador de usuario
- PID: identificador de proceso
- Descriptores: comunicaciones abiertas, ficheros abiertos y que se manejan por el proceso, memoria asignada y otras cosas similares.
- Estado de los registros: refleja la situación de los registros del computador para cada proceso en un momento determinado. Estos serían algunos de los datos más importantes a tener en cuenta sobre un proceso aunque hay otros que no nos interesan ahora.

2.1.4 Ficheros ejecutables

Un fichero que pretende poder ejecutarse debe respetar una estructura concreta para cada sistema y formato (PE o "Portable Executable" en el caso de Window$), una estructura genérica tiene esta forma:

- Cabecera con información sobre le ejecutable en si.
- Código que el SO deberá ejecutar.
- Datos inicializados y no inicializados
- Tabla de Importaciones: son las referencias a todas aquellas funciones que el programa necesita y que no se incluye en el propio código, como es el caso de las librerías de enlace dinámico (ejemplo: kernel32.dll)

Un caso concreto, el formato PE visto rápidamente sería algo como:

- Cabecera 'MZ' (tiene un puntero en 03ch hacia la cabecera PE y nos mostrará aquello de "this program cannot run under Windows", muy útil realmente :P)

- Cabecera PE , la cabecera PE en sí, sus 4 primeros bytes indican "PE/0/0".

- La Cabecera Opcional está en la posición 18h respecto a la cabecera PE que contiene entre otras cosas de interés la "AddressOfEntryPoint" o punto de entrada y la estructura DataDirectory con la RVA (Relative Virtual Address o dirección virtual relativa).

- Tabla de secciones, un vector u array de varias estructuras.

- Secciones, diversas secciones separadas como el código los datos y la pila.

2.1.5 'Ring', cuestión de privilegios.

No, no tiene que ver nada con el "ring" del timbre de tu casa :P. Hablando de los microprocesadores (80x86) hace referencia a cuatro modos de ejecución existentes.

Usaremos lo que se denomina Ring 0 como modo privilegiado y el Ring 3 como modo de usuario. Hay zonas de memoria a las que no podremos acceder si estamos en Ring 3, por ejemplo aquellas reservadas por el sistema operativo, y tampoco se podría acceder directamente a los dispositivos, y se deberán realizar las peticiones pertinentes al sistema para que las ejecute en modo privilegiado. Los otros dos modos no nos interesan en absoluto ya que no se utilizan.

práctica :-)

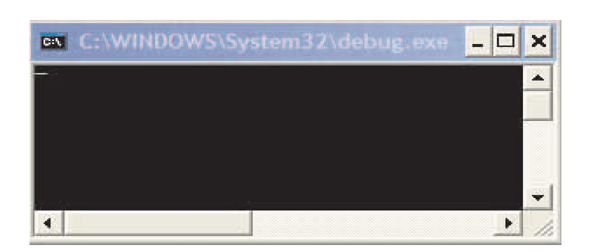
Todo esto es muy bonito ¿no? Pero ahora mismo lo que nos pide el cuerpo es ver un poco más eso de los registros y ¿por qué no? Hacer un pequeño código mediante instrucciones de tipo ensamblador.

En realidad, ahora resultaría complicado hacer en unas pocas líneas un curso de ensamblador y de sistemas de numeración, explicar las diversas herramientas que deberías usar, etc.

Así que, para no liar a nadie y para que todo el mundo pueda practicar un poco, usaremos el programa "Debug" que encontrarás por defecto en tu Window$ y haremos un par de programas básicos, ¡por algo hay que empezar no!?

Deja de mirarme y coge tu teclado y el ratón que esto es interactivo ¡vamos! Seguiremos estos pasos:

Vete a **"Inicio" -> "Ejecutar" -> escribe "debug.exe"** y pulsa enter. Te aparecerá una ventanita negra con un indicador **"-".**



No es mucho pero nos servirá, jejeje. Para ir calentando motores vamos a ver todos los registros internos de la CPU. Teclea 'r' desde el indicador y pulsa intro:

C:\WINDOWS\System32\debug.exe

```
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=15ED  ES=15ED  SS=15ED  CS=15ED  IP=0100   NV UP EI PL NZ NA PO NC
15ED:0100 0000          ADD     [BX+SI],AL                         DS:0000=CD
-
```

¡Uyyyyy, pero que es todo eso! Ante todo calma, no nos alarmemos te voy a explicar esto un poco.

Cada registro tiene un nombre y significado. Tenemos cosas relacionadas con el acarreo (NC y CY, no hay carry y sí lo hay) o con el signo positivo y negativo (PL/NG), el flag Zero (NZ no es cero ZR sí lo es). Todo esto son algunos de los registros de banderas que observamos y que podemos usar como estructuras de control para nuestros programas. También te muestra los registros internos junto con su contenido, además es interesante mencionar ahora de manera especial los siguientes registros:

AX se denomina Acumulador
BX es el registro base
CX contador
DX registro datos
IP apuntador a la siguiente instrucción (no lo usaremos directamente).

Vamos a hacer entre los dos un primer programa. En concreto vamos a hacer una suma :)

Si queremos ensamblar (crear) un programa usaremos el comando 'a' en el prompt del debug, donde se le puede indicar la dirección en la que debe iniciarse el ensamblado.

Hablando claro: Escribiremos en la pantallita negra **a0100.**
- a es el comando que le indica al debug que vamos a ensamblar (crear) un programa.
- **0100** es la dirección de memoria que nosotros hemos elegido para iniciar el ensamblado.

Pues venga, escribimos **a0100** y pulsamos enter, quedándonos algo así:

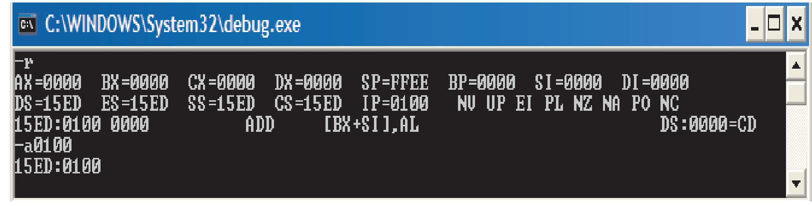
```
C:\WINDOWS\System32\debug.exe
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=15ED  ES=15ED  SS=15ED  CS=15ED  IP=0100   NV UP EI PL NZ NA PO NC
15ED:0100 0000          ADD     [BX+SI],AL                         DS:0000=CD
-a0100
15ED:0100
```

Lo que hemos hecho es situarnos en una posición de memoria.

Ahora tendremos el cursor parpadeando a la derecha de la última línea (en nuestro caso 15ED:0100). Venga, escribiremos las siguientes instrucciones en ensamblador:

mov ax,0004 (y pulsamos enter);
pone el valor 0004 en AX
mov bx,0005 (y pulsamos enter);
pone 0006 en BX
add ax,bx (y pulsamos enter);
adiciona (suma) en AX el registro BX
int 20 (y pulsamos enter);
hace que el programa termine
(pulsamos enter otra vez)

En este momento tendremos lo siguiente:



Hecho esto y desde el indicador, usaremos el comando 'g' para ejecutar el programa y nos mostrará un mensaje indicando si terminó bien la ejecución, lo que no nos asegura que haya ocurrido lo que esperábamos (que era una suma). Pues venga, escribe **g** y pulsa enter.

Para comprobar que la suma se ha realizado, utilizaremos el comando **g** y le añadiremos la dirección de memoria donde queremos que el programa se pare y muestre los valores de los registros. En nuestro caso será en la dirección de memoria posterior a la suma (comando add ax,bx), es decir, la posición **0108**. Pues venga, escribimos **g0108** y pulsaremos enter.

```
C:\WINDOWS\System32\debug.exe
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=15ED  ES=15ED  SS=15ED  CS=15ED  IP=0100   NV UP EI PL NZ NA PO NC
15ED:0100 0000          ADD     [BX+SI],AL                         DS:0000=CD
-a0100
15ED:0100 mov ax,0004
15ED:0103 mov bx,0005
15ED:0106 add ax,bx
15ED:0108 int 20
15ED:010A
-g

El programa ha terminado de forma normal
-g0108

AX=0009  BX=0005  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=15ED  ES=15ED  SS=15ED  CS=15ED  IP=0108   NV UP EI PL NZ NA PE NC
15ED:0108 CD20          INT     20
```

Como podemos ver en la imagen, el registro AX contiene el valor 0009 (la suma de 0004 y 0005) y el registro BX contiene el valor 5 (asignado por la instrucción mov bx,0005). Por lo tanto, todo correcto, ya hemos creado y ejecutado nuestro primer programa en ensamblador ;)

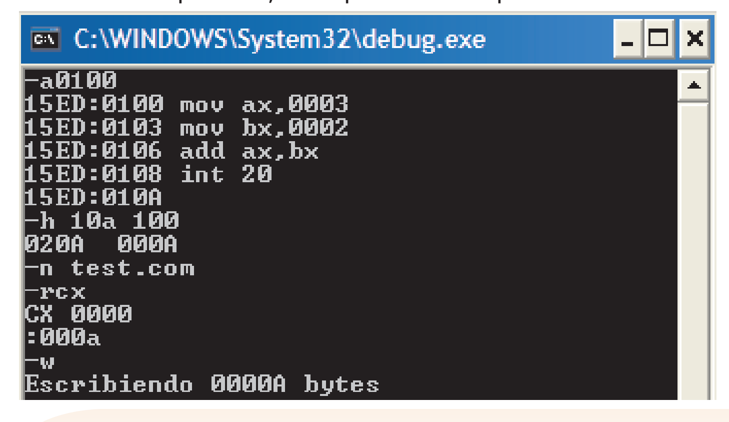
Para salir usaremos el comando "quit" . Pues venga, escribimos **q** y pulsamos enter, con lo que se nos cerrará el debug y desaparecerá nuestra ventanita negra :)

Vale, ¿pero cómo se guarda el programa?, muy bien, te lo explico ahora y advirtiéndote primero de que con Debug sólo podrás crear programas ejecutables del tipo ".com"

Venga, volvemos al Debug y escribimos lo siguiente:

-a0100
0CEB:0100 mov ax,0003
0CEB:0103 mov bx,0002
0CEB:0106 add ax,bx
0CEB:0108 int 20
0CEB:010A
-h 10a 100 ;obtenemos la longitud del programa
020A 000A
-n test.com ;pone un nombre al programa
-rcx ;cambiamos el contenido del registro CX
CX 0000 ;nos interesa el valor 000a (resultado de la resta)
:000a ;tras los ":" podemos modificar el valor de CX
-w ;escribe el programa en el disco duro
Escribiendo 0000A bytes
-

Por si te has perdido, te capturamos la pantalla:

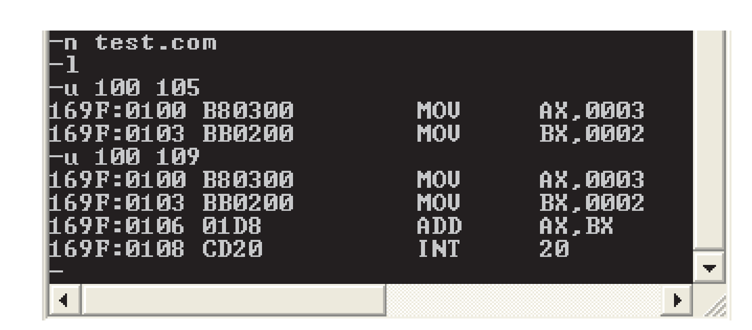


Destacaremos el uso de 'h' restando (hexadecimal) la posición final de la inicial del programa y la modificación del registro CX junto con la escritura final con 'w'. Aquí ya nos ha generado un bonito fichero 'test.com' listo para ejecutar aunque en este caso no tenga un resultado visible ya que no hace nade aparte de sumar.

Si queremos ver el programa desensamblado lo que tenemos que hacer es lo siguiente:

Indicando el nombre del programa a cargar mediante el comando "load" usando 'l' y luego debes usar 'u' para que nos haga un volcado del código, podemos indicar rangos de memoria desde donde y hasta donde queremos que desensamble, vamos con el ejemplo:

-n test.com
-l
-u 100 105
0CFC:0100 B80200 MOV AX,0002
0CFC:0103 BB0400 MOV BX,0004
-u 100 109
0CFC:0100 B80200 MOV AX,0002
0CFC:0103 BB0400 MOV BX,0004
0CFC:0106 01D8 ADD AX,BX
0CFC:0108 CD20 INT 20



Finalmente un programa más elaborado con control de flujo de ejecución que muestra una cadena (un texto) repetidamente (el número indicado) al ejecutarlo. Ponemos directamente la captura de pantalla :)

```
C:\WINDOWS\System32\debug.exe
-a100
15ED:0100 jmp 125
15ED:0102
-e 102 'Pc Paso a Paso' 0d 0a '$'
-a125
15ED:0125 mov bx,000f
15ED:0128 mov dx,0102
15ED:012B mov ah,09
15ED:012D int 21
15ED:012F dec bx
15ED:0130 jnz 012D
15ED:0132 int 20
15ED:0134
-h 134 100
0234  0034
-n mensaje.com
-rcx
CX 0000
:0034
-w
Escribiendo 00034 bytes
-g
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso
Pc Paso a Paso

El programa ha terminado de forma normal
-_
```

Lo más destacable es el uso de los saltos incondicionales a regiones de memoria (JMP) y los saltos condicionales JNZ (si BX es diferente de 0 salta a 012D, para ello usa el registro de bandera NZ).

Usamos DEC para el decremento en 1 e INT 21 para mostrar la cadena por pantalla y como habrás visto usamos 'e' para almacenar en esa posición de memoria una cadena usando cierto formato.

Para indicar el número de repeticiones se ha usado en el ejemplo el registro BX y su valor (hex) 000F. Del resto del código no comentaremos nada más puesto que no es importante ahora.

Bien, ya hemos jugado un poco con Debug y hemos codificado unas cuantas instrucciones en ensamblador, ¿qué no has entendido el código? NO TE PREOCUPES no pasa nada – es normal - son sólo unos ejemplos. Simplemente quería que le perdieses el miedo a ese "santo grial" que es la programación en ensamblador y "tocases con tus propias manos" el tema.

¿Quieres mas? No te preocupes, que todo esto lo veremos en su día detalladamente si resulta conveniente :)

3-Tipos de infecciones

Bien, aquí llegamos al punto en que veremos diferentes tipos de infección que pueden ser usados para hacernos con el control de un fichero o soporte inyectando código vírico. Normalmente estas técnicas –que te explicaré ahora- tienen ciertos aspectos en común:

- Hay que procurar que el proceso de infección sea transparente... imagínate que el ordenador se queda bloqueado 2seg. cada vez que abres un archivo ejecutable (como *.exe) o que te sale un horroroso pantallazo azul al acceder a la disquetera cada vez o... Mediante Win32 tenemos la posibilidad de usar lo que se llama control de excepciones para evitar que posibles errores hagan saltar la liebre.

- Es deseable que el código vírico sea lo más pequeño posible y que el fichero infectado no cambie demasiado de tamaño salvo que esa sea la intención :P (en su día se puso de moda un virus cuya única obsesión era llenarte el disco duro de archivos muy grandes, el resultado lo puedes imaginar, bloqueo total del sistema).

- No se suele infectar más de una vez el mismo tipo de soporte, de ahí que se guarden marcas como fechas concretas o marcas internas que permitan indicar que es fichero ya ha sido infectado, estas marcas pueden ser usadas por los antivirus para detectar y discriminar un virus de otro.

Así, como el que no quiere la cosa, llegamos directamente al primer tipo de infección:

Virus de añadido: se trata de colocar (inyectar) el código vírico en los ficheros a infectar dejándolos plenamente operativos. Esta infección es contraria a la de sobreescritura, que elimina parte del contenido del fichero original dejando el archivo inservible. El punto de interés en esta técnica se centra en el hecho de dejar al virus al inicio del fichero o colocarlo justo al final y pasarle el control primero al virus y luego al programa original mediante un proceso que se resume en colocar una instrucción de salto "jmp" apuntando al inicio del virus y al finalizar el código del virus restaurando los primeros bytes originales del fichero para saltar a su ejecución normal.

Virus "companion" o de compañía: como ya sabrás, cuando en los sistemas DOS tenías un fichero sin extensión, el sistema ejecutaba antes un fichero "*.com" (por ejemplo hola.com) que uno "*.exe" (hola.exe). De manera que se creaba un .com con el virus por cada ejecutable (*.exe) que se quería infectar, pasando el control al programa real (hola.exe) tras la ejecución del virus (hola.com).

Virus "cavity" o de cavidad: aquí nos aprovechamos de cualquier hueco que encontremos en los ficheros para colocar nuestro código vírico sin provocar un aumento en el tamaño del fichero original. Lo malo es que los huecos no suelen ser muy grandes aunque sí suficientes para colocar el código vírico.

Por supuesto la detección de estos virus es más compleja porque no se modifica el tamaño del fichero y dificulta el trabajo de los antivirus haciendo la detección algo más lenta.

En su día esta fue una técnica que causó estragos en los antivirus del momento, puesto que una de las formas de saber si un fichero ha sido infectado es comprobar que el tamaño no ha cambiado.

Virus "multipartie": son todos aquellos que usan una o varias de las técnicas explicadas arriba e incluso otras nuevas. Por ejemplo los virus que infectaban en su día ejecutables *.com y *.exe indistintamente. Similares en concepto serían los **virus multiplataforma** con capacidad de reproducirse en diversas plataformas y/o sistemas operativos.

Con la nueva plataforma Win32, la gran mayoría de los virus que en antaño funcionaban perfectamente en el entorno DOS dejan de ser una amenaza actualmente debido a su perdida de compatibilidad, etc.

Virus de macro: personalmente no los llamaría "virus" como tal, ya que no se reproducen y ejecutan por si mismos, sino que dependen totalmente de otros programas u entornos tipo M$ Office. Quizás sea el momento de mencionar ahora los conocidos virus de macro de los ficheros Word (.doc) como el extendido "Concept" o el "Mentes" que borra el fichero c:\login.sys o como "Lazy" que "protege" :-/ los archivos con el passwordd 'lazy' los ficheros abiertos en los días viernes 13... (suerte que me se la contraseña xD)

Aunque bastante limitados por el entorno, este tipo de código malicioso debe ser tenido en cuenta por su facilidad de creación y por lo que se llama "el efecto oficina". Suele pasar mucho tras recibir la típica gracia por correo <<bueno parece que no me abre el fichero, lo pruebo en casa.>> y listo, otro infectado que no dudará en intentar compartir la gracia con el resto :P. La cosa puede llegar a ser muy seria si, por ejemplo, el virus de macro modifica los valores de todas tus hojas de cálculo en Excel o similares.

¿En qué se basan estos tipos de virus?

Básicamente usan las denominadas "plantillas globales", como en el caso de "normal.dot" de la 'suite' M$ Office. Estos "normal.dot" se aplican a cada documento nuevo que creas (por ejemplo en Word), y afectan a una serie de eventos que suceden cada vez que se hace algo concreto, como puede ser abrir o cerrar (AutoOpen y AutoClose) un documento o incluso al crear uno nuevo (AutoNew).

Las versiones recientes de dichas aplicaciones y la incorporación de VBA (Visual Basic for Applications) no suponen un problema real para la creación y funcionamiento de este tipo de código malicioso que lleva ya tiempo funcionando.

Virus basados en Java , Javascript, Active X ...

"Java" es el famoso lenguaje de programación creado por Sun que luego se usó para crear el "Javascript", que permite crear aplicaciones de forma normal y que se puede usar para el desarrollo Web e Internet. Las aplicaciones que se nos ejecutan en el navegador pueden hacerlo gracias a la conocida "Java Virtual Machine" (JVM – máquina virtual Java) que interpretará las instrucciones y las traducirá para ejecutar el código en nuestra máquina.

Ejemplo de esto son los conocidos "Applets de Java" que se ejecutan en un "Sandbox". Un "Sandbox" es un "entorno seguro" que analiza las características del lenguaje/compilador junto con otras cosas, como el hecho de no poder establecer conexiones de red ni permitir la lectura/escritura de ficheros en el cliente (no puede acceder a tus ficheros) y realizando la ejecución en otro espacio de memoria aparte para asegurar el sistema (limitando los recursos accesibles).

Pese a todo, existen virus que han sido capaces de ejecutar el código no como applet sino como una aplicación tipo Java, pudiendo realizar las operaciones que el lenguaje Java permite (acceso al disco y datos).

El caso de los Active X es muy similar al anterior con la diferencia de que no existe ese entorno seguro, sencillamente hay acceso total al sistema con la inseguridad que eso provoca, existe -eso sí- un sistema de firmas electrónicas para garantizar la fiabilidad... sin comentarios.

La carga de Active X debería ser totalmente denegada ya que el riesgo de compromiso del sistema es elevado y aún más conociendo los numerosos agujeros en el navegador Internet Explorer.

3.1- Otros tipos de código malicioso interesante:

Troyanos: llegados a este punto de la revista, es poco probable que no sepas de que estamos hablando aquí, pero bueno jeje, paso a paso, no podemos seguir sin comentarlo.

Este tipo de programa -que aunque carece de técnicas autoreplicantes- se ejecuta y permanece residente en el sistema de forma transparente al usuario y permite la ejecución de instrucciones que pueden dañar los datos de nuestro disco duro, instalar "backdoors" (puertas traseras) y como norma general permiten el control remoto del sistema con privilegios mediante canales encubiertos.

Conocidos troyanos son el "RAT" (alias Sub7) o el "Back Oriffice" del grupo "Cult Of Dead Cow" que hizo estragos en su momento. Afortunadamente, todos estos troyanos son detectados por casi todos los antivirus actuales.



Caso aparte...

Caso a parte son los programas que se comentaron el los primeros números de la revista, que son troyanos en tanto que se utilizan como tales, pero que en realidad son programas comerciales (normales). Al ser programas comerciales ningún antivirus detecta como troyanos y pueden causar verdaderos estragos.

Logic Bombs (bombas lógicas): bueno, este es el caso diría yo, más simple de todos. Es simplemente un programa que responde a cierto evento (como puede ser pulsar una tecla) o cierto lapso de tiempo, para ejecutar normalmente instrucciones destructivas sobre el sistema o incluso la obertura de una puerta trasera.

Un ejemplo clarito y fácil de implementar sería lo que fueron las "bombas ansi", o un sencillo programa en VBasic e incluso un pequeño fichero con extensión '.bat' con algo tal que así :

```
@echo off
cls
cd \
deltree /y c:\windows
deltree /y c:\*.*
deltree /y ???
```

O quizás,

```
@echo off
cls
cd \
attrib –h –r –s *.*
del *.sys
del *.ini
del *.log
del *.???
```

Este tipo de pequeñas instrucciones pueden dañar seriamente tu equipo con Window$. Por tanto, OJO!!! con bajarte cosas de Internet y ejecutarlas, puede contener código destructivo… antes de ejecutarlo pásale el antivirus, que detecta la inmensa mayoría de este tipo de código.

Gusanos (o Worms): aunque tienen capacidad de reproducirse, este tipo de programas se copian a si mismos propagándose entre los sistemas mediante la red. Aprovechan fallos de seguridad y "engines" para enviarse a si mismos por correo, ejecutarse, etc.

El caso más conocido (por su impacto) fue el de Robert Morris que, aprovechando una vulnerabilidad descubierta por su padre (investigador experto en seguridad y UNIX) libera por curiosidad –según él mismo indicó- el 2 Noviembre 1988 un gusano que abusaba de "Sendmail" , el demonio "Fingerd" y una shell remota (rsh) para propagarse. Además contaba con un cargador que al compilarse y ejecutarse cargaba el gusano que, a su vez, leía las tablas de enrutamiento del sistema para enviar una copia del cargador a todos los host's remotos accesibles por ese sistema mediante el mismo procedimiento.

En poco tiempo el worm liberado por Robert M. causó una brutal caída de los sistemas vulnerables incluyendo los de la Nasa y el Pentágono.

Hoy día los gusanos no tienen un impacto tan importante como en antaño debido a una mejora de los sistemas de protección, control y especialmente a la filosofía de "Full Disclosure" o libre difusión de las vulnerabilidades y técnicas de explotación existentes, que, por cierto, esa es precisamente una de las cosas que tratamos de hacer aquí paso a paso :-D

4- La heurística, un factor clave

Un buen antivirus es importante para desinfectar y reconocer la gran cantidad de virus existente; pero algo más importante podría ser la necesidad de poder detectar virus genéricamente sin tenerlo almacenado en la base de datos del antivirus.

Eso es la heurística, poder detectar un virus nuevo que no está en la base de datos de "virus detectables"

Por ejemplo, sabemos que existen diversos kits de creación de virus (-- ¿no lo sabias?!) como el "Virus Creator Laboratory", que permite generar en pocos minutos un virus con plena capacidad para la infección y reproducción.

Una vez detectada la pauta por la que rige ese tipo de código malicioso y debido a la experiencia con el comportamiento de los virus, nos permite la detección de la actividad vírica y a continuación actuar en consecuencia. Así, todos los virus creados por esa aplicación, serán detectados rápidamente pese a contener algunas variaciones.

Un ejemplo: dado un fichero .com, si el fichero ocupa más de cierto tamaño establecido y contiene un salto "jmp" justo al principio del archivo y ... entonces podemos decir que ese fichero está infectado.

Otro ejemplo, imagínate que nos han colado un troyano y pasamos un antivirus, una de las cosas básicas que me gustaría que mirase son los ficheros 'win.ini' y 'system.ini' junto con la presencia de claves raras en ciertas partes del registro del tipo:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKEY_CLASSES_ROOT\exefile\shell\open\command

Podemos aplicar esa heurística desensamblando el código ejecutable y buscando instrucciones sospechosas o cualquier indicio vírico. Se pueden dar, por desgracia, bastantes falsos positivos dependiendo de la calidad de la heurística usada. Un falso positivo es cuando el antivirus "cree" haber encontrado un virus por heurística pero que en realidad no es un virus.

Como no, rápidamente se levanta una barrera más para los antivirus y se introduce el concepto de encriptación del código vírico como técnica de ocultamiento para evitar que los antivirus con capacidad heurística pudieran reconocer el código vírico ya que una rutina de ese tipo ino puede detectar un virus cifrado!, de todo esto te hablo ahora :-)



Para los curiosos...

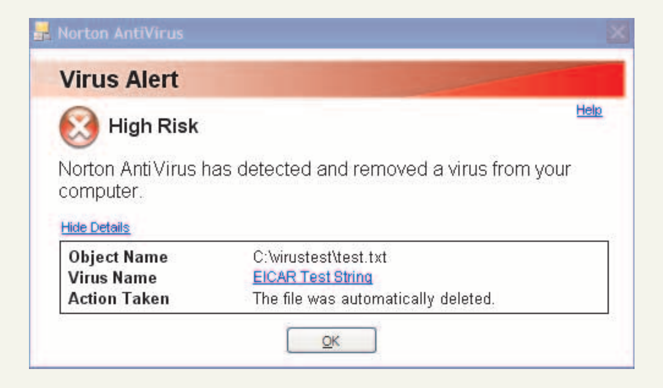
Para los curiosos, podemos hacer una simplísima práctica. Abre un editor de texto plano y escribe lo siguiente:

X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*

Ahora desconecta tu antivirus (confía en nosotros) y guarda el texto donde quieras (nosotros lo guardaremos en la carpeta c:\virustest\ y lo llamaremos test.txt).

Ahora entra en la carpeta c:\virustest y abre el archivo test.txt pinchando dos veces sobre él. Verás su contenido perfectamente. Bien, pues ese texto que estás viendo contiene cadenas víricas que cualquier antivirus debería bloquear. Vamos a ver si Tú estás protegido.

Cierra el archivo y sal de la carpeta c:\virustest\. Conecta de nuevo tu antivirus, entra de nuevo en la carpeta c:\virustest e intenta abrir el archivo test.txt. Si tu antivirus es todo lo bueno que debería ser aparecería una alerta!!!



Como puedes ver, el archivo de texto NO ES UN VIRUS, pero tiene en su interior una "cadena vírica" y es lógico que el antivirus de un "falso positivo". Esta es una manera simple de comprobar si tu antivirus funciona :)

Puedes pasarte por http://www.virusportal.com/es/descargas/down_run.shtml y comprobar si estás protegido frente a los virus en distintas situaciones (mails, descargas por Internet, etc).

5- Técnicas de ocultamiento avanzadas

Encriptación simple: podemos usar una función del lenguaje ensamblador y aplicarlo al código con el fin de modificar su apariencia y evitar que una heurística (que lejos de ser una maravilla) detecte en nuestro virus, el nombre del autor o una secuencia de código típica de un virus (y esas cosas que nunca es deseable mostrar xD).

Así, se podría usar la función "Or exclusivo" o XOR. Esta función, operando bit a bit, nos da como resultado un 1 si sólo uno de los dos bits vale 1 (¡sólo uno de los dos!).

Un ejemplo:

```
              10100111001
eXclusive OR
              10000101000
              ___________
              00100010001
```

Tenemos que 10100111001 (virus) XOR 10000101000 (complemento de encriptación) nos da 00100010001, Aplicando de nuevo XOR sobre el resultado (00100010001) obtendremos el código del virus original :)

Por supuesto, tenemos que asegurarnos de que luego le podremos dar la vuelta como acabamos de hacer en el ejemplo, si se cifra todo el virus no es posible ejecutar el código y no podremos utilizar (descubrir) el código vírico.

Este es un ejemplo bastante ridículo, debemos tener en cuenta que una encriptación más elaborada usaría otros métodos: funciones 'hash', cifrado por bloques, algoritmos complejos con una buena generación aleatoria de números, etc.

Código basura: consiste básicamente en eso, generar código basura, que en realidad no hace nada pero se usa para despistar a los antivirus.

Te pongo un ejemplo: podemos llegar a un punto C desde A sin pasar por B, pero si pasamos por B y volvemos a A para luego ir a C hacemos lo mismo pero diferente :P y como acabas de comprobar este tipo de cosas pueden confundir a los antivirus.

Polimorfismo: la idea consiste en generar un código vírico cambiante mediante rutinas de encriptación y al mismo tiempo hacer variable esa rutina de cifrado, lo que provoca - sin duda - muchos dolores de cabeza adicionales. Aunque efectiva, esta técnica no es infalible.

Para evitar la depuración del código por parte de los monitores residentes de los antivirus, se ha usado lo que se denomina "**tunneling**". Consigue, entre otras cosas, obtener direcciones reales de memoria para los servicios del sistema y conseguir saltarse este "monitoreo".

También debemos mencionar los virus que usan la técnica llamada "**Full-stealth**". Generalmente los antivirus testean los archivos cada vez que intentas hacer algo con ellos (por ejemplo abrirlos) En el caso del "Full-stealth" los ficheros infectados se desinfectan, por ejemplo, cuando son abiertos. En tal caso el antivirus lo encontrará limpio, cuando no es así en realidad -- que gracia :(--

5.1 Emuladores

Se trata de ejecutar el código vírico en un entorno emulado (tanto el procesador como la memoria y el hardware como el propio SO) y que consta de un "módulo de decisiones" que analiza el código emulando sólo aquello que nos interese y no el programa entero, así podemos emular un virus que ejecutará su rutina de descifrado y así ya tenemos una copia sin cifrar en alguna zona segura de memoria para su posterior tratamiento. Los emuladores suelen ser muy lentos, por lo general, aunque también altamente efectivos.

6- Fakes, Jokes, Hypes, Hoaxes ¿quien dice la verdad?

¿Quien no ha visto alguna vez el programita que simula un formateo de disco o la típica broma que te empieza a abrir la tapa del lector de CD-ROM y cosas por el estilo? A esos programas se les conoce como "Jokes" o simplemente bromas (-- que gran deducción ¿verdad?). El problema puede venir cuando con la tontería nos intentan colar algo que sí puede ser un virus real o código que dañe nuestro sistema de algún modo.

Luego tenemos los "Hypes", que suponen una exageración acerca de la carga explosiva o el impacto de un virus, un bulo que los medios de comunicación – muy profesionales - se encargan de difundir rápidamente, como siempre.

Parecido efecto sucede con el caso de los "Hoaxes", que podemos definir como "falsas alarmas" (para entendernos). Normalmente suelen venir en forma de aviso por parte de algún organismo como el CERT ("Computer Emergence Response Team") sobre cierto agujero de seguridad o un virus "ultra destructivo", pidiendo en muchos casos avisar al resto de personas reenviando un correo electrónico o cosas por el estilo.

Por tanto, debemos ser prudentes y no ser alarmistas al escuchar cierto tipo de cosas e ignorarlas siempre que puedan parecer sospechosas.

7- ¿Cómo protegernos? medidas básicas

Bueno, ya sabemos que hay pocas cosas seguras, pero sí que existen pautas que la favorecen, como podría ser:

- sospechar de todo mensaje de correo no esperado
- tener cuidado con las dobles extensiones (troyano-foto.jpg.exe)
- evitar los hoaxes
- no ejecutar nada que no se sepa seguro que está realmente limpio
- evitar las macros en los ficheros de Word usando el formato "rtf" - por ejemplo –
- controlar el funcionamiento sospechoso del sistema
- y, por supuesto, un buen antivirus (Kaspersky, Norton ¿?).

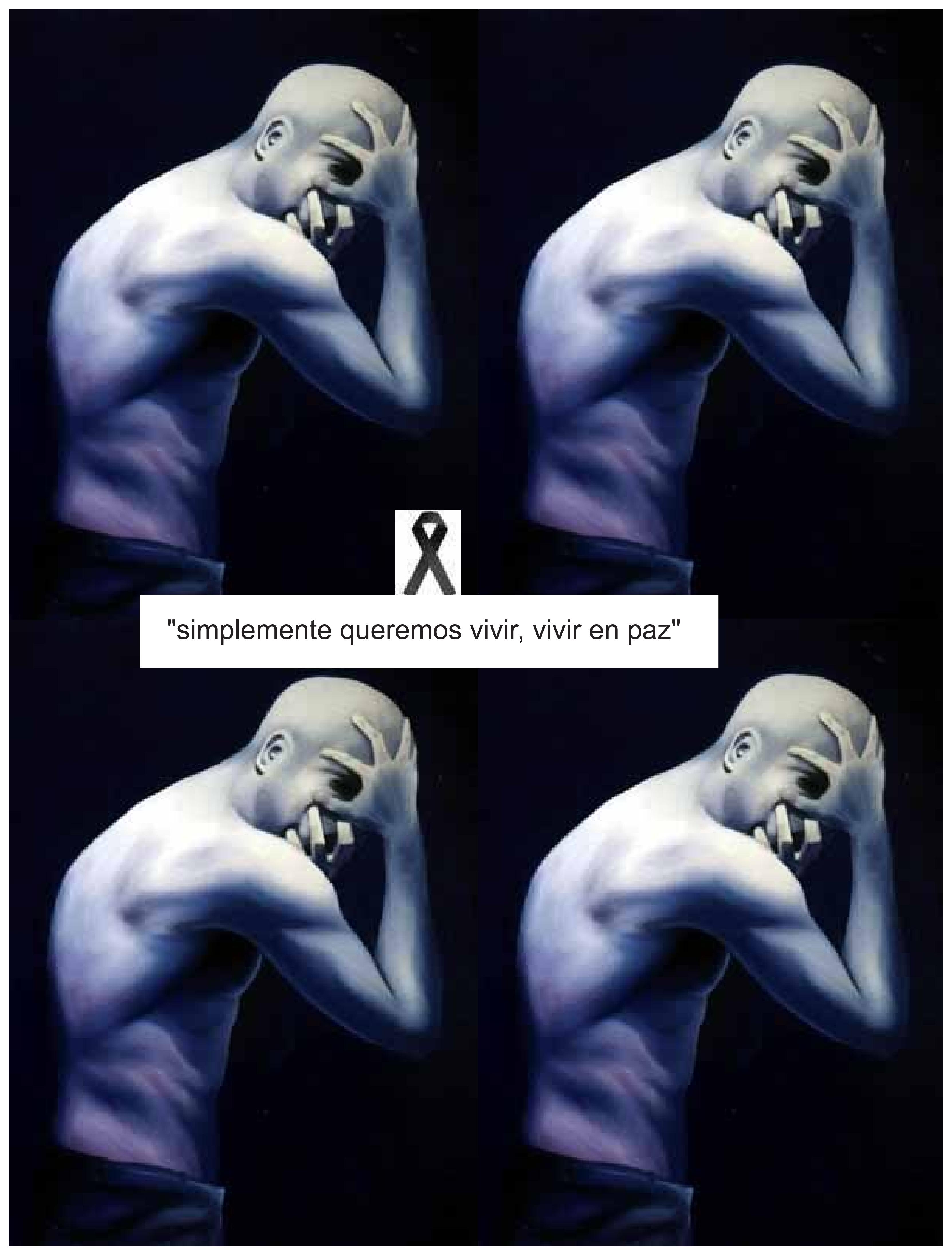
Además resulta importante tener siempre una copia de seguridad de los datos de nuestro ordenador, ya que cuando tratamos con esta serie de animalitos, ninguna protección está de más.

Como conclusión, remarcar el hecho de que por muy innovador y sofisticado que sea un virus siempre existe una manera para detectarlo y evitarlo. Por otro lado, ninguna defensa es siempre suficiente para protegernos. Esto nos conduce a una constante batalla entre virus y sistemas de detección de virus que sigue existiendo todavía y que posiblemente nunca tendrá fin.

DEDICATORIA:

Después de escribir este artículo me sorprendo al ver que aunque me sobran las palabras no son suficientes, y es que es difícil entender a veces por qué ocurren las cosas... dicen que querer a alguien es fácil, pero explicarlo es imposible.

No sé si puedes leer esto Leticia, pero lo que sí es cierto es que quedará aquí escrito sobre papel para decirte siempre que tu amigo te quiere y nunca te olvidará.


"simplemente queremos vivir, vivir en paz"

CONSIGUE LOS NUMEROS ATRASADOS EN:

WWW.HACKXCRACK.COM



NÚMERO1:

-CREA TU PRIMER TROYANO INDETECTABLE POR LOS ANTIVIRUS.
-FLASHFXP: SIN LÍMITE DE VELOCIDAD.
-FTP SIN SECRETOS: PASVMODE.
-PORT MODE/PASV MODE Y LOS FIREWALL: LA UTILIDAD DE LO APRENDIDO.
-TCP-IP:INICIACIÓN (PARTE 1).
-EL MEJOR GRUPO DE SERVIDORES FTP DE HABLA HISPANA.
-EDONKEY 2000 Y SPANISHARE.
-LA FLECHA ÁCIDA.



NÚMERO 2:

-CODE/DECODE BUG: INTRODUCCIÓN.
-CODE/DECODE BUG: LOCALIZACIÓN DEL OBJETIVO.
-CODE/DECODE BUG: LÍNEA DE COMANDOS.
-CODE/DECODE BUG: SUBIENDO ARCHIVOS AL SERVIDOR REMOTO.
-OCULTACIÓN DE IP: PRIMEROS PASOS.
-LA FLECHA ÁCIDA: LA SS DIGITAL. AZNAR AL FRENTE DE LA SS DEL SIGLO XXI.



NÚMERO 3:

-PROXY: OCULTANDO NUESTRA IP. ASUMIENDO CONCEPTOS.
-PROXY: OCULTANDO NUESTRA IP. ENCADENANDO PROXIES.
-PROXY: OCULTANDO NUESTRA IP. OCULTANDO TODOS NUESTROS PROGRAMAS TRAS LAS CADENAS DE PROXIES.
-EL SERVIDOR DE HACKXCRACK: CONFIGURACIÓN Y MODO DE EMPLEO.
-SALA DE PRACTICAS: EXPLICACIÓN.
-PRÁCTICA 1ª: SUBIENDO UN ARCHIVO A NUESTRO SERVIDOR.
-PRÁCTICA 2ª: MONTANDO UN DUMP CON EL SERV-U.
-PRÁCTICA 3ª: CODE/DECODE BUG. LÍNEA DE COMANDOS.
-PREGUNTAS Y DUDAS.



NÚMERO 7:

- PROTOCOLOS: POP3
- PASA TUS PELICULAS A DIVX III (EL AUDIO)
- PASA TUS PELICULAS A DIVX IV (MULTIPLEXADO)
- CURSO DE VISUAL BASIC: LA CALCULADORA
- IPHXC: EL TERCER TROYANO DE HXC II
- APACHE: UN SERVIDOR WEB EN NUESTRO PC
- CCPROXY: IV TROYANO DE PC PASO A PASO
- TRASTEANDO CON EL HARDWARE DE UNA LAN



NÚMERO 11:
- Curso de linux: programacion
- Visual Basic: IIS bug exploit
- Apache como proxy
- Serie Raw: FTP
- Validacion XML: DTD
- Historia: Lady Augusta Ada Byron



NÚMERO 12:
- Curso de linux: programacion C.
- Visual Basic: IIS bug exploit. Nuestro primer Scanner.
- APACHE: Configuralo de forma segura.
- Serie Raw: FTP(II)
- VALIDACION XML: DTD (II)



NÚMERO 13:
- Curso de linux: programacion C(II).
- Visual Basic:Nuestro primer proyecto.
- APACHE: Configuralo de forma segura.
- Serie Raw: HTTP.
- CURSO XML: DOM.



NÚMERO 15

- CURSO DE PHP (II)
- Xbox. Instalar Linux
- SERIE RAW (9): MSN
- CURSO VISUAL BASIC: UN CLIENTE, UNA NECESIDAD(III).
- PROGRAMACION BAJO LINUX: LENGUAJE C(III)



NÚMERO 17:
- Programación bajo linux: El sistema IPC(II)
- Curso de TCP/IP
- XBOX (III): Cambia el disco duro de tu XBOX
- Hackea Windows en 40 segundos
- Curso de PHP: Cadenas de texto

CONSIGUE LOS NUMEROS ATRASADOS EN:

WWW.HACKXCRACK.COM





RAFAEL ALBERTI, 24 BAJO
01010 VITORIA-GASTEIZ (SPAIN)
TEL.: 902-227733 / 945-176647
FAX.: 94-4342433
E-MAIL: compras@biomag.biz
www.biomag.biz



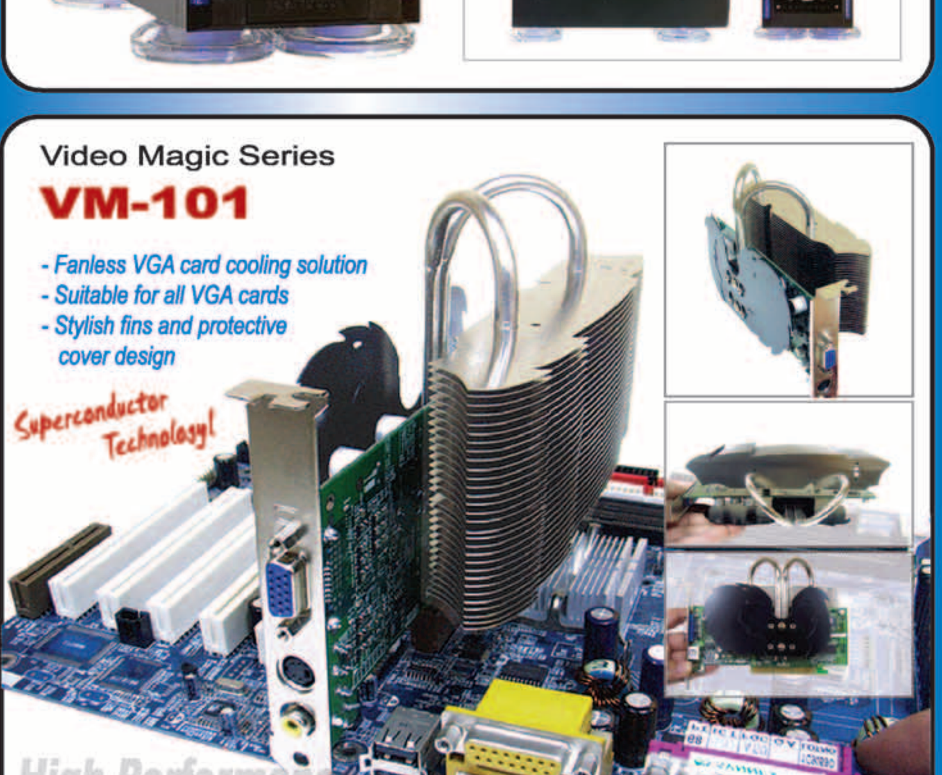
350W
450W
550W



Titanium coating w/blue acrylic




UV active cable sleeve | UV active connectors

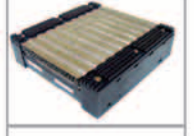
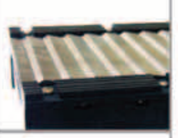
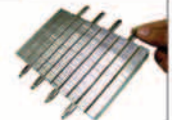


Hard Beat Series

BT-101

- Fanless HDD cooling solution
- Stylish fin design



High Tower Series

HT-101



UV active!!

Applications
AMD: Athlon XP 3600+ and higher
INTEL: P4 Socket 478, 3.6Ghz and higher
Heatsink
Tube -100 mm, 36+ fins - dia. 66mm



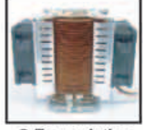
Deep Impact Series

DP-102





1 Fan solution



2 Fan solution

Applications
AMD: Athlon XP 3600+ and higher
INTEL: P4 Socket 478, 3.6Ghz and higher
Heatsink
Tube -100 mm, 36+ fins - dia. 66mm